N/A
Internet-Draft
Intended status: Informational
Expires: March 15, 2013

T. Keiser
Sine Nomine
S. Jenkins

A. Deason, Ed.
Sine Nomine

September 11, 2012

AFSVol Tag-Length-Value Remote Procedure Call Extensions
draft-tkeiser-afs3-volser-tlv-04

Abstract

AFS-3 is a distributed file system based upon prototypes developed at Carnegie Mellon University during the 1980s. AFS-3 heavily leverages Remote Procedure Calls (RPCs) as the foundation for its distributed architecture. This memo extends the volume management interface to support getting and setting of AFS volume attributes via an extensible Tag-Length-Value (TLV) encoding, which is based upon AFS-3 extensible discriminated unions. TLV-based get and set RPCs are specified, along with a tag enumeration RPC.

In addition, tags are allocated for existing volume and transaction metadata, and implementation-private tags are allocated for metadata related to the OpenAFS Demand Attach File Server, and the RxOSD protocol suite.

Internet Draft Comments

Comments regarding this draft are solicited. Please include the AFS-3 protocol standardization mailing list (afs3-standardization@openafs.org) as a recipient of any comments.

AFS-3 Document State

This document is in state "draft", as per the document state definitions set forth in [I-D.wilkinson-afs3-standardisation].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

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

working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

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

Copyright Notice

Table of Contents

1. Introduction

AFS-3 [CMU-ITC-88-062] [CMU-ITC-87-068] is a distributed file system that has its origins in the VICE project [CMU-ITC-84-020] [CMU-ITC-85-039] at the Carnegie Mellon University Information Technology Center [CMU-ITC-83-025] a joint venture between CMU and IBM. VICE later became AFS when CMU moved development to a new commercial venture called Transarc Corporation, which later became IBM Pittsburgh Labs. AFS-3 is a suite of un-standardized network protocols based on a remote procedure call (RPC) suite known as Rx. While de jure standards for AFS-3 fail to exist, the various AFS-3 implementations have agreed upon certain de facto standards, largely helped by the existence of an open source fork called OpenAFS that has served the role of reference implementation. In addition to using OpenAFS as a reference, IBM wrote and donated developer documentation that contains somewhat outdated specifications for the Rx protocol and all AFS-3 remote procedure calls, as well as a detailed description of the AFS-3 system architecture.

The AFS-3 architecture consists of many administrative domains called "cells" [CMU-ITC-88-070] which are glued together to form a globally distributed file system. Each cell consists of: client nodes, which run cache manager daemons; file servers, which run file server daemons and volume server daemons; and database server nodes, which can run volume location database servers, protection database servers, backup database servers, or several other obscure and/or deprecated database services.

This memo focuses on the volume server [AFS3-VVL] component of AFS-3. The volume server provides an RPC interface for managing AFS volumes. Volumes are the unit of storage administration in AFS-3. Each volume contains a subtree of the file system, along with special directory entries called mount points, which are used to link volumes together into a (potentially cyclic) directed graph. Mount points can cross cell boundaries, thus permitting construction of a cross-organizational, globally distributed, location-transparent file system. The file system is location-transparent because mount points contain volume names and cell names (which are resolved to locations by contacting the appropriate cell's volume location database), rather than encoding the data's physical location directly in the pointer. This memo extends the AFS-3 volume server RPC interface with a suite of new RPCs that provide extensible volume metadata get and set operations.

1.1. Motivations

The current AFSVol volume metadata introspection routines use hard-coded XDR [RFC4506] structure definitions. This significantly limits

protocol extensibility because new remote procedure calls and structure definitions must be defined during each protocol revision. To some degree, this has been due to the lack of protocol standards documents: certain sites co-opted unused protocol fields for private uses, thus precluding the standards process from reclaiming these fields (without breaking existing deployments). Hence, each time new functionality is required, a new RPC--and, typically, new XDR data structures--need to be defined. This is a rather expensive process--both in terms of standardization, and implementation. Frequently, this leads to a desire to postpone protocol feature enhancements until many changes can be aggregated into a monolithic protocol upgrade.

This memo introduces a new tag-length-value (TLV) encoding mechanism based upon the AFS-3 extensible discriminated union primitive type [I-D.keiser-afs3-xdr-union]. This TLV encoding is utilized for getting and setting AFS-3 volume metadata. The key advantage of this design is that new TLV tuples can be allocated without defining a new RPC. Furthermore, because ext-union includes a length field in the encoding, it is always possible for a peer to decode the remainder of the XDR stream--even when a tag (and, thus, its XDR encoding) is unrecognized. Hence, decode error handling can happen at the application layer, instead of deep within the Rx protocol stack.

As the TLV changes require the addition of several new RPC interfaces (that are intended to eventually supplant extant interfaces), it is logical to add AFSVol capabilities [I-D.keiser-afs3-capabilities] for these new interfaces.

1.2. Goals

This memo aims to standardize a new TLV encoding mechanism for volume metadata. In addition, this memo will standardize the TLV encoding of volume metadata which is currently available via several AFSVol XDR structures, as well as specify the encoding of several new pieces of AFS-3 volume metadata that are not currently available via the AFSVol interface. For example, metadata specific to the OpenAFS Demand Attach File Server [DAFS] will be made available via the AFSVol service, whereas in the past it was only available locally on the file server machine via a proprietary interprocess communication mechanism.

1.3. Abbreviations

AFS - Historically, AFS stood for the Andrew File System; AFS no longer stands for anything

afscbint - AFS-3 Cache Manager RPC Interface Definition

afsint - AFS-3 File Server RPC Interface Definition

AFSVol - AFS Volume Server Rx RPC Implementation

CM - AFS-3 Cache Manager

DAFS - OpenAFS Demand Attach File Server

FS - AFS-3 File Server

RPC - Remote Procedure Call

RPC-L - Rx RPC Interface Definition Language (fork of ONC RPC [RFC5531] .x file format)

Rx - The Remote Procedure Call mechanism utilized by AFS-3 [AFS3-RX]

RXAFS - AFS File Server Rx RPC Implementation

RXAFSCB - AFS Cache Manager Rx RPC Implementation

TLV - Tag-Length-Value encoding

TSV - Tag nameSpace Version ordinal

TTL - Time to Live for cached data

volint - AFS-3 Volume Server RPC Interface Definition

volser - AFS-3 Volume Server

XDR - eXternal Data Representation [RFC4506]

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. New Capability Bit Allocations

3.1. VICED_CAPABILITY_DAFS

When this capability bit is asserted, the file server is advertising that it supports Demand Attach File Server version 1 protocol semantics. Specifically, DAFS v1 semantics imply that the following invariants MAY be violated by the fileserver:

1. any indication, which is visible to an afsint client, that a file server is restarting implies that all call back state will be invalidated; and

2. RXAFS_GetVolumeStatus will return exact on-disk header state for the volume in question.

3.2. AFSVOL_CAPABILITY_DAFS

When this capability bit is asserted, the volserver is advertising that it supports Demand Attach File Server version 1 protocol semantics. Specifically, DAFS v1 semantics imply that the following invariants MAY be violated by the volserver:

1. RPCs returning volintInfo (AFSVolListVolumes, AFSVolListOneVolume) and volintXInfo (AFSVolXListVolumes, AFSVolXListOneVolume) will return exact on-disk header state for the volume in question

In addition, the combination of AFSVOL_CAPABILITY_DAFS and AFSVOL_CAPABILITY_TLV MAY imply that the tag AFSVOL_TLV_TAG_VOL_STATE_DAFS_RAW exists. However, this implication SHOULD NOT be relied upon, as DAFS may evolve to the point where AFSVOL_TLV_TAG_VOL_STATE_DAFS_RAW has to be deprecated. When both of these capabilities are asserted, the client SHOULD still gracefully handle the VOLSER_TAG_UNSUPPORTED error for AFSVOL_TLV_TAG_VOL_STATE_DAFS_RAW.

3.3. AFSVOL_CAPABILITY_TLV

Assertion of this capability bit indicates the ability to service the RPC calls described in Section 4.

4. TLV Interface

A new suite of RPCs will be standardized to get/set tag-length-value tuples, and to enumerate supported tags. The tag namespace will be controlled by the AFS Assigned Numbers Registrar as an assigned

numbers namespace.

4.1. Encoding

The TLV data will be encoded using the AFS-3 extensible discriminated union [I-D.keiser-afs3-xdr-union]. The RPC-L specification for the base TLV types is as follows (NB: this is pseudocode; a less-concise, but machine-parseable version of this specification is included in Appendix A).

```
/* registrar-controlled tag namespace */
enum AFSVol_TLV_tag {
    ...
};

const AFSVOL_TLV_TAG_MAX = 1024;          /* upper-bound on number of
                                           * TLV tuples per RPC */
const AFSVOL_TLV_OPAQUE_MAX = 262144;     /* upper-bound on size of
                                           * value payload */
const AFSVOL_TLV_TIME_MAX = 21845;        /* upper-bound on length of
                                             AFSTime vector payload */
const AFSVOL_TLV_UINT64_MAX = 32768;      /* upper-bound on length of
                                             uint64 vector payload */

enum AFSVol_TLV_type {
    AFSVOL_TLV_TYPE_NULL          = 0,
    AFSVOL_TLV_TYPE_TRUE          = 1,
    AFSVOL_TLV_TYPE_FALSE         = 2,
    AFSVOL_TLV_TYPE_UINT64        = 3,
    AFSVOL_TLV_TYPE_UINT64_VEC    = 4,
    AFSVOL_TLV_TYPE_INT64         = 5,
    AFSVOL_TLV_TYPE_INT64_VEC     = 6,
    AFSVOL_TLV_TYPE_UUID          = 7,
    AFSVOL_TLV_TYPE_STRING        = 8,
    AFSVOL_TLV_TYPE_TIME_ABS      = 9,
    AFSVOL_TLV_TYPE_TIME_ABS_VEC  = 10,
    AFSVOL_TLV_TYPE_TIME_REL      = 11,
    AFSVOL_TLV_TYPE_TIME_REL_VEC  = 12,
    AFSVOL_TLV_TYPE_VOL_ID        = 13,
    AFSVOL_TLV_TYPE_VOL_ID_VEC    = 14,
    AFSVOL_TLV_TYPE_PART_ID       = 15,
    AFSVOL_TLV_TYPE_PART_ID_VEC   = 16,
    AFSVOL_TLV_TYPE_DISK_BLOCKS   = 17,
    AFSVOL_TLV_TYPE_STAT_COUNTER  = 18,
    AFSVOL_TLV_TYPE_STAT_GAUGE    = 19,
    AFSVOL_TLV_TYPE_BIT64         = 20,
    AFSVOL_TLV_TYPE_VOL_DOW_USE   = 21,
    AFSVOL_TLV_TYPE_OPAQUE        = 22
```

```
   };

   ext-union AFSVol_TLV_value switch(AFSVol_TLV_type type) {
    case AFSVOL_TLV_TYPE_NULL:
    case AFSVOL_TLV_TYPE_TRUE:
    case AFSVOL_TLV_TYPE_FALSE:
       void;

    case AFSVOL_TLV_TYPE_UINT64:
    case AFSVOL_TLV_TYPE_VOL_ID:
    case AFSVOL_TLV_TYPE_PART_ID:
    case AFSVOL_TLV_TYPE_DISK_BLOCKS:
    case AFSVOL_TLV_TYPE_STAT_COUNTER:
    case AFSVOL_TLV_TYPE_BIT64:
       afs_uint64 u_u64;

    case AFSVOL_TLV_TYPE_INT64:
    case AFSVOL_TLV_TYPE_STAT_GAUGE:
       afs_int64 u_s64;

    case AFSVOL_TLV_TYPE_UINT64_VEC:
    case AFSVOL_TLV_TYPE_VOL_ID_VEC:
    case AFSVOL_TLV_TYPE_PART_ID_VEC:
       afs_uint64 u_u64_vec<AFSVOL_TLV_UINT64_MAX>;

    case AFSVOL_TLV_TYPE_INT64_VEC:
       afs_int64 u_s64_vec<AFSVOL_TLV_UINT64_MAX>;

    case AFSVOL_TLV_TYPE_TIME_ABS:
       AFSTime u_time_abs;

    case AFSVOL_TLV_TYPE_TIME_REL:
       AFSRelTimestamp u_time_rel;

    case AFSVOL_TLV_TYPE_TIME_ABS_VEC:
       AFSTime u_time_abs_vec<AFSVOL_TLV_TIME_MAX>;

    case AFSVOL_TLV_TYPE_TIME_REL_VEC:
       AFSRelTimestamp u_time_rel_vec<AFSVOL_TLV_UINT64_MAX>;

    case AFSVOL_TLV_TYPE_UUID:
       afsUUID u_uuid;

    case AFSVOL_TLV_TYPE_STRING:
       string u_string<AFSVOL_TLV_OPAQUE_MAX>;

    case AFSVOL_TLV_TYPE_VOL_DOW_USE:
       /* type defined later in this memo */
```

```
        AFSVol_stat_use_per_dow u_vol_dow_use;

     case AFSVOL_TLV_TYPE_OPAQUE:
        opaque u_opaque<AFSVOL_TLV_OPAQUE_MAX>;
    };

    const AFSVOL_TLV_FLAG_UNSUPPORTED = 0x1;
    const AFSVOL_TLV_FLAG_READ_ERROR = 0x2;
    const AFSVOL_TLV_FLAG_CRITICAL = 0x4;
    const AFSVOL_TLV_FLAG_QUALIFIER_NO_MATCH = 0x8;
    const AFSVOL_TLV_FLAG_MORE = 0x10;
    const AFSVOL_TLV_FLAG_OBJ_NOT_SUPP = 0x20;

    struct AFSVol_TLV {
        afs_uint32 tlv_tag;
        afs_uint32 tlv_flags;
        AFSVol_TLV_value tlv_value;
    };
```

TLV XDR pseudocode

Figure 1

4.1.1. Data Value Types

The core of the TLV definition above is the AFS-3 extensible discriminated union primitive type. The following discriminators are initially defined in this memo:

AFSVOL_TLV_TYPE_NULL = 0

This shall map to type XDR void in the AFSVol_TLV_value union.

AFSVOL_TLV_TYPE_TRUE = 1

This shall map to type XDR void in the AFSVol_TLV_value union. It is used to communicate the boolean value true.

AFSVOL_TLV_TYPE_FALSE = 2

This shall map to type XDR void in the AFSVol_TLV_value union. It is used to communicate the boolean value false.

AFSVOL_TLV_TYPE_UINT64 = 3

This shall map to type afs_uint64 in the AFSVol_TLV_value union. The semantics of this field are defined by the tag.

AFSVOL_TLV_TYPE_UINT64_VEC = 4

This shall map to an XDR variable length vector of up to 32768 afs_uint64 values. The semantics of this field are defined by the tag.

AFSVOL_TLV_TYPE_INT64 = 5

This shall map to type afs_int64 in the AFSVol_TLV_value union. The semantics of this field are defined by the tag.

AFSVOL_TLV_TYPE_INT64_VEC = 6

This shall map to an XDR variable length vector of up to 32768 afs_int64 values. The semantics of this field are defined by the tag.

AFSVOL_TLV_TYPE_UUID = 7

This shall map to an afsUUID [I-D.keiser-afs3-xdr-primitive-types] type.

AFSVOL_TLV_TYPE_STRING = 8

This shall map to an XDR string of maximum length 262144. The semantics of this field are defined by the tag.

AFSVOL_TLV_TYPE_TIME_ABS = 9

This shall map to an AFSTime in the AFSVol_TLV_value union. This absolute timestamp shall be encoded using the rules specified in the AFS-3 time type specification [I-D.deason-afs3-type-time].

AFSVOL_TLV_TYPE_TIME_ABS_VEC = 10

This shall map to an XDR variable length vector of up to 21845 AFSTime values in the AFSVol_TLV_value union. The absolute timestamp contained within each vector element shall be encoded using the rules specified in the AFS-3 time type specification [I-D.deason-afs3-type-time].

AFSVOL_TLV_TYPE_TIME_REL = 11

This shall map to an AFSRelTimestamp in the AFSVol_TLV_value union. This relative timestamp (time interval) shall be encoded using the rules specified in the AFS-3 time type specification [I-D.deason-afs3-type-time].

AFSVOL_TLV_TYPE_TIME_REL_VEC = 12

This shall map to an XDR variable length vector of up to 32768 AFSRelTimestamp values in the AFSVol_TLV_value union. The relative timestamp (time interval) contained within each vector element shall be encoded using the rules specified in the AFS-3 time type specification [I-D.deason-afs3-type-time].

AFSVOL_TLV_TYPE_VOL_ID = 13

This shall map to an afs_uint64 in the AFSVol_TLV_value union. This field shall contain an AFS-3 volume identifier. When transmitting 32-bit volume identifiers, the upper 32 bits of this field MUST all be zeroes.

AFSVOL_TLV_TYPE_VOL_ID_VEC = 14

This shall map to an XDR variable length vector of up to 32768 afs_uint64 values in the AFSVol_TLV_value union. The elements within this vector shall contain AFS-3 volume identifiers. When transmitting 32-bit volume identifiers, the upper 32 bits of the value MUST all be zeroes.

AFSVOL_TLV_TYPE_PART_ID = 15

This shall map to an afs_uint64 in the AFSVol_TLV_value union. This field shall contain an AFS-3 vice partition identifier. When transmitting 32-bit partition identifiers, the upper 32 bits of this field MUST all be zeroes.

AFSVOL_TLV_TYPE_PART_ID_VEC = 16

This shall map to an XDR variable length vector of up to 32768 afs_uint64 values in the AFSVol_TLV_value union. The elements within this vector shall contain AFS-3 vice partition identifiers. When transmitting 32-bit partition identifiers, the upper 32 bits of the value MUST all be zeroes.

AFSVOL_TLV_TYPE_DISK_BLOCKS = 17

This shall map to an afs_uint64 in the AFSVol_TLV_value union. This field shall contain an unsigned integer count in units of (1024 octet) disk blocks.

AFSVOL_TLV_TYPE_STAT_COUNTER = 18

This shall map to an afs_uint64 in the AFSVol_TLV_value union. This field shall contain an unsigned integer counter statistic.

AFSVOL_TLV_TYPE_STAT_GAUGE = 19

This shall map to an afs_int64 in the AFSVol_TLV_value union. This field shall contain a signed integer gauge (level) statistic.

AFSVOL_TLV_TYPE_BIT64 = 20

This shall map to an afs_uint64 in the AFSVol_TLV_value union. This field shall contain a 64-bit bit field.

AFSVOL_TLV_TYPE_VOL_DOW_USE = 21

This shall map to an type AFSVol_stat_use_per_dow, as defined in Section 6.4.

AFSVOL_TLV_TYPE_OPAQUE = 22

This shall map to an XDR opaque byte array of maximum length 262144. The semantics and encoding of this field are defined by the tag (please see Section 4.1.3 for further discussion).

4.1.2. TLV Flags

The AFSVol_TLV structure contains a 32-bit flags field for communication of various ancillary boolean values. This memo defines and allocates the following flag bits:

AFSVOL_TLV_FLAG_UNSUPPORTED = 0x1

When this flag is asserted, it tells the RPC caller that this tag is not supported by this server.

AFSVOL_TLV_FLAG_READ_ERROR = 0x2

When this flag is asserted, it tells the RPC caller that the server was unable to read a value for this tag, despite the tag being supported by the server.

AFSVOL_TLV_FLAG_CRITICAL = 0x4

When this flag is asserted, it informs the peer that failure to decode the payload associated with this tag is a fatal error that should result in aborting this RPC call.

AFSVOL_TLV_FLAG_QUALIFIER_NO_MATCH = 0x8

When this flag is asserted, it informs the caller that the qualifier passed in did not match any record.

AFSVOL_TLV_FLAG_MORE = 0x10

When this flag is asserted, it informs the caller that the server was unable to send all available tags because the AFSVOL_TLV_TAG_MAX XDR vector length limit was exceeded.

AFSVOL_TLV_FLAG_OBJ_NOT_SUPP = 0x20

When this flag is asserted, it tells the RPC caller that the object in question does not support this specific tag, despite the tag generally being supported by the server. An example of this scenario could be a volume object which was created using a format too old to support new metadata fields.

4.1.3. Use of AFSVOL_TLV_TYPE_OPAQUE

When possible, future protocol augmentations requiring the definition of new data types should request allocation of a new standards-track payload type code. Allocation of a type code should coincide with standardization of the payload encoding associated with the type code allocation. However, in limited circumstances where:

1. it is known a priori that there will never be any encoding ambiguity (i.e., the opaque type for this tag shall map to exactly one XDR type), and

2. the cost of type code allocation and encoding standardization are deemed too high

use of the type code AFSVOL_TLV_TYPE_OPAQUE may be an acceptable alternative.

4.2. Qualifiers

In some cases the value associated with a tag will be large, structured data. A qualifier is a tag-specific parameter which allows a caller to address a subset of the value stored in a tag. For TLV get interfaces, specifying a qualifer can reduce the amount of data sent over the wire. For TLV set interfaces, specifying a qualifier permits a client to modify a subset of a structured value without endangering cache coherence. Qualifiers are marshalled over the wire as type AFSVol_TLV_value. Unless otherwise noted, it should be assumed that a tag only supports the null qualifier (ext-union

discriminator set to AFSVOL_TLV_TYPE_NULL). The null qualifier always references the entire value for a given tag.

5. AFSVol TLV Interface

5.1. Error Codes

A collection of new wire error codes are a required substrate. The following new error codes are defined:

VOLSER_TAG_UNSUPPORTED

This server implementation does not support this tag.

VOLSER_TAG_READ_ONLY

This tag is marked read-only, and thus AFSVolSetVolumeTLV cannot be invoked.

VOLSER_TAG_WRITE_FAILED

AFSVolSetVolumeTLV failed to write to this tag for an unspecified reason.

VOLSER_TAG_DECODE_FAILED

AFSVolSetVolumeTLV could not set the value passed for this tag because the AFSVol_TLV_value ext-union decoder raised an error flag. This error may also be returned if, e.g., the XDR type includes opaque data, then failure to validate the octet stream embedded within the opaque would also result in this error code.

VOLSER_TAG_UNSUPPORTED_ENCODING

The discriminant passed in AFSVol_TLV_value to AFSVolSetVolumeTLV is not supported by this implementation for this particular tag (e.g., an AFSVOL_TLV_TYPE_OPAQUE value was passed to a tag which only supports AFSVOL_TLV_TYPE_UINT64).

VOLSER_TLV_QUALIFIER_DECODE_FAILED

A qualifier could not be decoded because an AFSVol_TLV_value ext-union decoder raised an error flag. This error may also be returned if, e.g., the XDR type includes opaque data, then failure to validate the octet stream embedded within the opaque would also result in this error code.

VOLSER_TLV_QUALIFIER_UNSUPPORTED_ENCODING

The discriminant passed in AFSVol_TLV_value in the qualifier is not supported by this implementation for this particular tag (e.g., an AFSVOL_TLV_TYPE_OPAQUE value was passed as the qualifier to a tag which only supports qualifiers of type AFSVOL_TLV_TYPE_UINT64).

VOLSER_TLV_QUALIFIER_INVALID

The qualifier passed to the RPC was successfully decoded, and was of a supported type, however the value of the qualifier failed application-layer sanity checks.

VOLSER_TRANS_INVALID

An RPC failed to execute because the transaction ID passed as an IN parameter was invalid.

5.2. Tag Introspection

In order for clients to determine which tags are supported by a given server, an RPC is provided for obtaining the list of tags. In addition to returning the tags, this RPC also returns a tag namespace version (TSV) ordinal that can be used by clients to determine whether a consistent set of tags was fetched from the server. Additionally, this version ordinal can be compared against the value returned by all other TLV RPC calls to determine whether the tag cache remains coherent. The Rx procedure specification for the tag enumeration RPC is as follows:

```
typedef afs_uint64 AFSVol_TLV_TSV;
typedef AFSVol_TLV_tag AFSVol_TLV_tag_vec<AFSVOL_TLV_TAG_MAX>;

proc GetVolumeTLVTags(
    IN AFSVol_TLV_tag offset,
    OUT AFSVol_TLV_tag_vec * tags,
    OUT AFSVol_TLV_TSV * tsv
) = XXX;
```

Figure 2

A compliant implementation MUST implement this RPC. The call parameters are defined as follows:

offset

The offset IN parameter specifies the numeric offset of the first tag to return. A value of zero indicates that the client wants to start the enumeration at the beginning of the tag list.

tags

The tags OUT parameter contains a sorted list of supported tags, beginning with the first supported tag greater than or equal to the offset IN parameter.

tsv

The tsv OUT parameter contains the current server tag namespace version ordinal. When this value changes from the previously-seen value on the client, it indicates that tag namespace cache coherence has been lost, and the client SHOULD restart fetching the tag namespace from offset zero.

5.2.1. Tag Namespace Cache Coherence

Because the AFSVol interface is stateless, cache coherence cannot be maintained via the normal AFS mechanism. Thus, AFSVol clients MUST treat enumerated tags as ephemeral with a TTL of two hours.

The Tag nameSpace Version (TSV) ordinal is used to communicate the current version of the tag namespace to the caller. Version zero is a reserved value, so the server must initialize this verion ordinal to a value great than 1. Servers MUST ensure that version ordinals are unique within any 2 hour TTL period.

5.3. TLV Get

The Rx procedure specification for the TLV get interface will be as follows:

```
struct AFSVol_TLV_query {
    AFSVol_TLV_tag tq_tag;
    AFSVol_TLV_value tq_qualifier;
};

typedef AFSVol_TLV_query AFSVol_TLV_query_vec<AFSVOL_TLV_TAG_MAX>;
typedef AFSVol_TLV AFSVol_TLV_vec<AFSVOL_TLV_TAG_MAX>;

proc GetOneVolumeTLV(
    IN afs_uint64 partId,
    IN afs_uint64 volId,
    IN AFSVol_TLV_query_vec * queries,
    OUT AFSVol_TLV_vec * tuples,
    OUT AFSVol_TLV_TSV * tsv
) = XXX;
```

Figure 3

A compliant implementation MUST implement this RPC. The call parameters are defined as follows:

partId

The partId IN parameter specifies the disk partition on which the volume is located.

volId

The volId IN parameter specifies the volume for which TLV tuples are being requested.

queries

The queries IN parameter specifies an optional list of tags for which TLV tuples are desired. If this parameter is zero-length, then the server will return up to AFSVOL_TLV_TAG_MAX TLV tuples. If an unknown tag identifier is passed in the tags parameter, then the server will return a tuple with the AFSVOL_TLV_FLAG_UNSUPPORTED bit asserted in AFSVol_TLV.tlv_flags, and the tlv type set to AFSVOL_TLV_TYPE_NULL. If the volume object in question does not support a given metadata tag, then a tuple will be returned with AFSVOL_TLV_FLAG_OBJ_NOT_SUPP set in the AFSVol_TLV.tlv_flags field, and the tlv type set to AFSVOL_TLV_TYPE_NULL. Similarly, if the server is unable to retrieve the value for a supported tag, then a tuple will be returned with AFSVOL_TLV_FLAG_READ_ERROR set in the AFSVol_TLV.tlv_flags field, and the tlv type set to AFSVOL_TLV_TYPE_NULL. The AFSVol_TLV_query.tq_qualifier field

contains optional tag-specific qualifiers which would allow the implementation to return a subset of the data for a specific tag. When a non-NULL qualifier is passed, and the qualifier fails to match any record, then the flag bit AFSVOL_TLV_FLAG_QUALIFIER_NO_MATCH will be set in AFSVol_TLV.tlv_flags field, and the tlv type set to AFSVOL_TLV_TYPE_NULL.

tuples

The tuples OUT parameter contains up to AFSVOL_TLV_TAG_MAX TLV tuples for this volume. If all tags cannot be sent due to AFSVOL_TLV_TAG_MAX vector length limit, then the flag bit AFSVOL_TLV_FLAG_MORE SHALL be asserted in the last element.

tsv

The tsv OUT parameter contains the current server tag namespace version ordinal. When this value changes from the previously-seen value on the client, it indicates that tag namespace cache coherence has been lost, and the client SHOULD use AFSVolGetVolumeTLVTags (see Figure 2) to re-fetch the tag namespace.

5.4. TLV Streaming Get

This call is similar to the call described in the previous section, with the exception that TLV tuples will be returned for multiple volumes at once using an Rx split call interface. The Rx procedure specification is as follows:

```
const AFSVOL_BULK_GETVOLUME_MAX = 1024;

typedef afs_uint64 AFSVol_TLV_part_id_vec<AFSVOL_BULK_GETVOLUME_MAX>;
typedef afs_uint64 AFSVol_TLV_vol_id_vec<AFSVOL_BULK_GETVOLUME_MAX>;

struct AFSVol_TLV_vol_list {
    afs_uint64 partId;
    AFSVol_TLV_Vol_id_vec * volIds;
};

typedef struct AFSVol_TLV_vol_list
AFSVol_TLV_get_filter<AFSVOL_BULK_GETVOLUME_MAX>;

proc GetVolumesTLV(
    IN AFSVol_TLV_get_filter * filter,
    IN AFSVol_TLV_query_vec * queries,
    OUT AFSVol_TLV_TSV * tsv
) split = XXX;
```

Figure 4

Implementation of this RPC is OPTIONAL. The call parameters are defined as follows:

filter

The filter IN parameter specifies an optional list of vice partitions and volume identifiers. If the filter vector is zero-length, then TLV information is requested for all volumes on all vice partitions. If this list is non-zero in length, then TLV information is requested only for volumes on specific vice partitions. Furthermore, if the volIds vector in a specific filter vector index is non-zero in length, then TLV information is only returned for thoese volumes whose identifiers are enumerated in the volIds vector.

queries

The queries IN parameter specifies an optional list of tags for which TLV tuples are desired. If this parameter is zero-length, then the server will return up to AFSVOL_TLV_TAG_MAX TLV tuples. If an unknown tag identifier is passed in the tags parameter, then the server will return a tuple with the AFSVOL_TLV_FLAG_UNSUPPORTED bit asserted in AFSVol_TLV.tlv_flags, and the tlv type set to AFSVOL_TLV_TYPE_NULL. If the volume object in question does not support a given metadata tag, then a tuple will be returned with AFSVOL_TLV_FLAG_OBJ_NOT_SUPP set in the AFSVol_TLV.tlv_flags field, and the tlv type set to

AFSVOL_TLV_TYPE_NULL. Similarly, if the server is unable to retrieve the value for a supported tag, then a tuple will be returned with AFSVOL_TLV_FLAG_READ_ERROR set in the AFSVol_TLV.tlv_flags field, and the tlv type set to AFSVOL_TLV_TYPE_NULL. The AFSVol_TLV_query.tq_qualifier field contains optional tag-specific qualifiers which would allow the implementation to return a subset of the data for a specific tag. When a non-NULL qualifier is passed, and the qualifier fails to match any record, then the flag bit AFSVOL_TLV_FLAG_QUALIFIER_NO_MATCH will be set in AFSVol_TLV.tlv_flags field, and the tlv type set to AFSVOL_TLV_TYPE_NULL.

tsv

The tsv OUT parameter contains the current server tag namespace version ordinal. When this value changes from the previously-seen value on the client, it indicates that tag namespace cache coherence has been lost, and the client SHOULD use AFSVolGetVolumeTLVTags (see Figure 2) to re-fetch the tag namespace.

5.4.1. Split call stream encoding

The contents of the split call stream shall be an xdrrec stream containing a finite sequence of XDR-encoded AFSVol_TLV structures, each of which shall be marked as a separate record (typically by calling xdrrec_endofrecord). End of sequence will be annotated by a dummy tuple containing the special tag type AFSVOL_TLV_TAG_EOS.

5.5. TLV Set

The Rx procedure specification for the TLV set interface will be as follows:

```
struct AFSVol_TLV_store {
    AFSVol_TLV ts_tuple;
    AFSVol_TLV_value ts_qualifier;
};

typedef AFSVol_TLV_store AFSVol_TLV_store_vec<AFSVOL_TLV_TAG_MAX>;
typedef afs_int32 AFSVol_TLV_result_vec<AFSVOL_TLV_TAG_MAX>;

proc SetVolumeTLV(
    IN afs_int32 trans,
    IN AFSVol_TLV_TSV assert_tsv,
    IN AFSVol_TLV_store_vec * tuples,
    OUT AFSVol_TLV_result_vec * results,
    OUT AFSVol_TLV_TSV * server_tsv
) = XXX;
```

Figure 5

Implementation of this RPC is OPTIONAL. The call parameters are defined as follows:

trans

The trans IN parameter specifies the transaction ID returned by a previous invocation of AFSVolTransCreate.

assert_tsv

The assert_tsv IN parameter contains the TSV ordinal expected by the client. When this parameter is zero, it implies that the client wants the values to be set, regardless of the current server tag namespace version. However, if the value of this parameter is non-zero, it MUST match the current server TSV. When there is a TSV mismatch, the call MUST fail with error code VOLSER_TAG_TSV_MISMATCH.

tuples

The tuples IN parameter contains the list of TLV tuples to be set by the server.

results

The results OUT parameter contains a list of error codes, one per tuple. These error codes provide specific information regarding the success/failure of each TLV set operation. Valid error codes include:

* VOLSER_TAG_UNSUPPORTED

* VOLSER_TAG_READ_ONLY

* VOLSER_TAG_WRITE_FAILED

* VOLSER_TAG_DECODE_FAILED

* VOLSER_TAG_UNSUPPORTED_ENCODING

* VOLSER_TAG_TSV_MISMATCH

* VOLSER_TLV_QUALIFIER_UNSUPPORTED_ENCODING

* VOLSER_TLV_QUALIFIER_DECODE_FAILED

* VOLSER_TLV_QUALIFIER_INVALID

* VOLSERFAILEDOP

* VOLSERBAD_ACCESS

* VOLSER_TRANS_INVALID

server_tsv

The tsv OUT parameter contains the current server tag namespace version ordinal. When this value changes from the previously-seen value on the client, it indicates that tag namespace cache coherence has been lost, and the client SHOULD use AFSVolGetVolumeTLVTags (see Figure 2) to re-fetch the tag namespace.

5.5.1. Call preprocessing

The SetVolumeTLV begins by scanning all elements within the tuples array. If any elements have the AFSVOL_TLV_FLAG_CRITICAL bit asserted in tuples[i].ts_tuple.ts_flags, then preprocessing of the tuple must occur. For each tuple with the critical bit set, several preprocessing validation steps will be taken.

5.5.1.1. Verify tag is supported

The tag stored in tuples[i].ts_tuple.tlv_tag is checked to ensure that the server supports it. In the event that the tag is not supported, then the corresponding array index in the results array will be set to VOLSER_TAG_UNSUPPORTED, and the RPC call abort at the conclusion of critical tuple preprocessing with error code

VOLSERFAILEDOP.

5.5.1.2. Verify tag is writeable

The tag stored in tuples[i].ts_tuple.tlv_flag is checked to ensure that it is a writeable property. In the event that the tag is read-only, then the corresponding array index in the results array will be set to VOLSER_TAG_READ_ONLY, and the RPC call will abort at the conclusion of critical tuple preprocessing with error code VOLSERFAILEDOP.

5.5.1.3. Verify value encoding is supported

The ext-union discriminator in tuples[i].ts_tuple.tlv_value is checked to make sure that it is a supported type. If the discriminator is not a supported type, then the corresponding array index in the results array will be set to VOLSER_TAG_UNSUPPORTED_ENCODING, and the RPC call will abort at the conclusion of critical tuple preprocessing with error code VOLSERFAILEDOP.

5.5.1.4. Verify value can be decoded

The value stored in tuples[i].ts_tuple.tlv_value is checked to make sure that it can be decoded. If the wire-encoded data cannot be decoded, then the corresponding array index in the results array will be set to VOLSER_TAG_DECODE_FAILED, and the RPC call will abort at the conclusion of critical tuple preprocessing with error code VOLSERFAILEDOP.

5.5.1.5. Verify qualifier is supported

Qualifiers are specific to a given tag. If for any reason the tag-specific validation logic determines that the qualifier is invalid, it may set the corresponding array index in the results array to one of VOLSER_TLV_QUALIFIER_UNSUPPORTED_ENCODING, VOLSER_TLV_QUALIFIER_DECODE_FAILED, or VOLSER_TLV_QUALIFIER_INVALID. As with the other validation steps, if a critical tuple fails qualifier validation, then the RPC call will abort at the conclusion of critical tuple preprocessing with error code VOLSERFAILEDOP.

5.5.2. Call processing

Once the necessary validation steps have been performed, the call will perform the set operations for each tuple. Errors encountered during the processing of each tuple will be recorded in the appropriate array index of the results array. At the conclusion the RPC will either return 0 if all set operations succeeded, or

VOLSERFAILEDOP if any failed.

6. Mapping of existing metadata onto TLV namespace

Existing metadata available from several interfaces will also be exported as TLV tuples. This is being done not only for completeness, but also to prevent data races between AFSVolGetOneVolumeTLV, and the various legacy introspection interfaces.

6.1. volintXInfo

All metadata exported via the volintXInfo XDR structure will now be exported as TLV tuples. Unless otherwise specified, the values associated with each tag shall be identical to that returned for the associated field in volintXInfo by the AFSVolXListOneVolume interface. The following tuples will be allocated to export existing members of volintXInfo:

AFSVOL_TLV_TAG_VOL_NAME

This is the TLV analogue of volintXInfo.name. This tuple MUST have a payload of type AFSVOL_TLV_TYPE_STRING. The u_string payload field MUST contain a null-terminated string.

AFSVOL_TLV_TAG_VOL_STATUS

This is the TLV analogue of volintXInfo.status. This tuple MUST have payload of type AFSVOL_TLV_TYPE_UINT64.

AFSVOL_TLV_TAG_VOL_IN_USE

This is the TLV analogue of volintXInfo.inUse. This tuple will contain a boolean value, and therefore MUST have a payload type of either: AFSVOL_TLV_TYPE_TRUE, or AFSVOL_TLV_TYPE_FALSE.

AFSVOL_TLV_TAG_VOL_ID

This is the TLV analogue of volintXInfo.volid. This tuple MUST have a payload of type AFSVOL_TLV_TYPE_VOL_ID.

AFSVOL_TLV_TAG_VOL_TYPE

This is the TLV analogue of volintXInfo.type. This tuple MUST have a payload of type AFSVOL_TLV_TYPE_UINT64.

AFSVOL_TLV_TAG_VOL_CLONE_ID

This is the TLV analogue of volintXInfo.cloneID. This tuple MUST have a payload of type AFSVOL_TLV_TYPE_VOL_ID.

AFSVOL_TLV_TAG_VOL_BACKUP_ID

This is the TLV analogue of volintXInfo.backupID. This tuple MUST have a payload of type AFSVOL_TLV_TYPE_VOL_ID.

AFSVOL_TLV_TAG_VOL_PARENT_ID

This is the TLV analogue of volintXInfo.parentID. This tuple MUST have payload of type AFSVOL_TLV_TYPE_VOL_ID.

AFSVOL_TLV_TAG_VOL_COPY_DATE

This is the TLV analogue of volintXInfo.copyDate. This tuple MUST have payload of type AFSVOL_TLV_TYPE_TIME_ABS.

AFSVOL_TLV_TAG_VOL_CREATE_DATE

This is the TLV analogue of volintXInfo.creationDate. This tuple MUST have payload of type AFSVOL_TLV_TYPE_TIME_ABS. This timestamp shall be encoded using the rules specified in the AFS-3 time type specification [I-D.deason-afs3-type-time].

AFSVOL_TLV_TAG_VOL_ACCESS_DATE

This is the TLV analogue of volintXInfo.accessDate. This tuple MUST have payload of type AFSVOL_TLV_TYPE_TIME_ABS. This timestamp shall be encoded using the rules specified in the AFS-3 time type specification [I-D.deason-afs3-type-time].

AFSVOL_TLV_TAG_VOL_UPDATE_DATE

This is the TLV analogue of volintXInfo.updateDate. This tuple MUST have payload of type AFSVOL_TLV_TYPE_TIME_ABS. This timestamp shall be encoded using the rules specified in the AFS-3 time type specification [I-D.deason-afs3-type-time].

AFSVOL_TLV_TAG_VOL_BACKUP_DATE

This is the TLV analogue of volintXInfo.backupDate. This tuple MUST have payload of type AFSVOL_TLV_TYPE_TIME_ABS. This timestamp shall be encoded using the rules specified in the AFS-3 time type specification [I-D.deason-afs3-type-time].

AFSVOL_TLV_TAG_VOL_SIZE

This is the TLV analogue of volintXInfo.size. This tuple MUST have payload of type AFSVOL_TLV_TYPE_DISK_BLOCKS.

AFSVOL_TLV_TAG_VOL_FILE_COUNT

This is the TLV analogue of volintXInfo.filecount. This tuple MUST have payload of type AFSVOL_TLV_TYPE_STAT_GAUGE.

AFSVOL_TLV_TAG_VOL_QUOTA_BLOCKS

This is the TLV analogue of volintXInfo.maxquota. This tuple MUST have payload of type AFSVOL_TLV_TYPE_DISK_BLOCKS.

AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY

This is the TLV analogue of volintXInfo.dayUse. This tuple MUST have payload of type AFSVOL_TLV_TYPE_STAT_COUNTER. This field tracks volume accesses by AFS-3 clients over the course of this calendar day, since midnight local time of the file server.

Operational monitoring applications which need to correlate the start time for the counter against a date SHOULD simultaneously query the value of tag AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY_DATE. For further discussion of the cache coherence implications, please see Section 6.4.2.

It should be noted that the definition of an "access" is implementation-private, and thus comparison of access rates across AFS-3 implementations is not possible.

AFSVOL_TLV_TAG_VOL_STAT_USE_PER_DOW

This is the TLV exportation of the daily usage statistics for the past week. This tuple may have two different payload types, depending upon whether or not a qualifier is delivered. The payload and qualifier types will be discussed in Section 6.4.

It should be noted that the definition of an "access" is implementation-private, and thus comparison of access rates across AFS-3 implementations is not possible.

AFSVOL_TLV_TAG_VOL_STAT_READS

This is the TLV analogue of volintXInfo.stat_reads. This tuple MUST have payload of type AFSVOL_TLV_TYPE_UINT64_VEC. This vector SHALL be of length 4.

AFSVOL_TLV_TAG_VOL_STAT_WRITES

This is the TLV analogue of volintXInfo.stat_reads. This tuple MUST have payload of type AFSVOL_TLV_TYPE_UINT64_VEC. This vector SHALL be of length 4.

AFSVOL_TLV_TAG_VOL_STAT_FILE_SAME_AUTHOR

This is the TLV analogue of volintXInfo.stat_fileSameAuthor. This tuple MUST have payload of type AFSVOL_TLV_TYPE_UINT64_VEC. This vector SHALL be of length 6.

AFSVOL_TLV_TAG_VOL_STAT_FILE_DIFFERENT_AUTHOR

This is the TLV analogue of volintXInfo.stat_fileDiffAuthor. This tuple MUST have payload of type AFSVOL_TLV_TYPE_UINT64_VEC. This vector SHALL be of length 6.

AFSVOL_TLV_TAG_VOL_STAT_DIR_SAME_AUTHOR

This is the TLV analogue of volintXInfo.stat_dirSameAuthor. This tuple MUST have payload of type AFSVOL_TLV_TYPE_UINT64_VEC. This vector SHALL be of length 6.

AFSVOL_TLV_TAG_VOL_STAT_DIR_DIFFERENT_AUTHOR

This is the TLV analogue of volintXInfo.stat_dirDiffAuthor. This tuple MUST have payload of type AFSVOL_TLV_TYPE_UINT64_VEC. This vector SHALL be of length 6.

6.2. transDebugInfo

All metadata exported via the transDebugInfo XDR structure will now be exported as TLV tuples. Unless otherwise specified, the values associated with each tag shall be identical to that returned for the associated field in transDebugInfo by the AFSVolMonitor interface. The following tuples will be allocated to export existing members of transDebugInfo:

AFSVOL_TLV_TAG_VOL_TRANS_ID

This is the TLV analogue of transDebugInfo.tid. This tuple MUST have payload of type AFSVOL_TLV_TYPE_UINT64.

AFSVOL_TLV_TAG_VOL_TRANS_TIME

This is the TLV analogue of transDebugInfo.time. This tuple MUST have payload of type AFSVOL_TLV_TYPE_TIME_REL.

AFSVOL_TLV_TAG_VOL_TRANS_CREATE_TIME

This is the TLV analogue of transDebugInfo.creationTime. This tuple MUST have payload of type AFSVOL_TLV_TYPE_TIME_ABS.

AFSVOL_TLV_TAG_VOL_TRANS_RETURN_CODE

This is the TLV analogue of transDebugInfo.returnCode. This tuple MUST have payload of type AFSVOL_TLV_TYPE_INT64.

AFSVOL_TLV_TAG_VOL_TRANS_ATTACH_MODE

This is the TLV analogue of transDebugInfo.iflags. This tuple MUST have payload of type AFSVOL_TLV_TYPE_BIT64.

AFSVOL_TLV_TAG_VOL_TRANS_STATUS

This is the TLV analogue of transDebugInfo.vflags This tuple MUST have payload of type AFSVOL_TLV_TYPE_BIT64.

AFSVOL_TLV_TAG_VOL_TRANS_FLAGS

This is the TLV analogue of transDebugInfo.tflags. This tuple MUST have payload of type AFSVOL_TLV_TYPE_BIT64.

AFSVOL_TLV_TAG_VOL_TRANS_LAST_PROC_NAME

This is the TLV analogue of transDebugInfo.lastProcName. This tuple MUST have payload of type AFSVOL_TLV_TYPE_STRING. The u_string payload field MUST contain a null-terminated string.

AFSVOL_TLV_TAG_VOL_TRANS_CALL_VALID

This is the TLV analogue of transDebugInfo.callValid. This tuple will contain a boolean value, and therefore MUST have a payload type of either: AFSVOL_TLV_TYPE_TRUE, or AFSVOL_TLV_TYPE_FALSE.

AFSVOL_TLV_TAG_VOL_TRANS_READ_NEXT

This is the TLV analogue of transDebugInfo.readNext. This tuple MUST have payload of type AFSVOL_TLV_TYPE_STAT_COUNTER. This field contains the next expected Rx data packet sequence number expected by the receive side of this transaction's bulk data transfer operation.

AFSVOL_TLV_TAG_VOL_TRANS_XMIT_NEXT

This is the TLV analogue of transDebugInfo.transmitNext. This tuple MUST have payload of type AFSVOL_TLV_TYPE_STAT_COUNTER. This field contains the next Rx data packet sequence number to be used by the transmit side of this transaction's bulk data transfer operation.

AFSVOL_TLV_TAG_VOL_TRANS_LAST_RECV_TIME

This is the TLV analogue of transDebugInfo.lastReceiveTime. This tuple MUST have payload of type AFSVOL_TLV_TYPE_TIME_ABS.

AFSVOL_TLV_TAG_VOL_TRANS_LAST_SEND_TIME

This is the TLV analogue of transDebugInfo.lastSendTime. This tuple MUST have payload of type AFSVOL_TLV_TYPE_TIME_ABS.

6.3. Additional de facto-standardized fields

Certain fields from the IBM AFS and OpenAFS file server's VolumeDiskData header are generally useful. In particular, several fields exported via the AFSVolGetFlags and AFSVolSetFlags RPCs should be exported via the TLV interface. The full list of supported TLV tuples are:

AFSVOL_TLV_TAG_VOL_IN_SERVICE

This tuple will contain a boolean value, and therefore MUST have a payload type of either: AFSVOL_TLV_TYPE_TRUE, or AFSVOL_TLV_TYPE_FALSE. When this bit is not asserted, the volume is administratively prohibited from coming online.

AFSVOL_TLV_TAG_VOL_BLESSED

This tuple will contain a boolean value, and therefore MUST have a payload type of either: AFSVOL_TLV_TYPE_TRUE, or AFSVOL_TLV_TYPE_FALSE. When this bit is not asserted, the volume is administratively prohibited from coming online.

AFSVOL_TLV_TAG_VOL_RESTORED_FROM_ID

This tuple MUST have payload of type AFSVOL_TLV_TYPE_VOL_ID. When this field is non-zero, it contains the volume ID contained in the dump from which it was restored.

AFSVOL_TLV_TAG_VOL_DESTROYED

This tuple will contain a boolean value, and therefore MUST have a payload type of either: AFSVOL_TLV_TYPE_TRUE, or AFSVOL_TLV_TYPE_FALSE. When this bit is asserted, this volume is flagged for deletion.

AFSVOL_TLV_TAG_VOL_NEEDS_SALVAGE

This tuple will contain a boolean value, and therefore MUST have a payload type of either: AFSVOL_TLV_TYPE_TRUE, or AFSVOL_TLV_TYPE_FALSE. When this bit is asserted, this volume requires a salvage.

AFSVOL_TLV_TAG_VOL_OFFLINE_MESSAGE

This tuple MUST have payload of type AFSVOL_TLV_TYPE_STRING. The u_string payload field MUST contain a null-terminated string. This field stores an administrative message to indicate why the volume is offline.

AFSVOL_TLV_TAG_VOL_EXPIRATION_DATE

This tuple MUST have payload of type AFSVOL_TLV_TYPE_TIME_ABS. This timestamp shall be encoded using the rules specified in the AFS-3 time type specification [I-D.deason-afs3-type-time]. To the best knowledge of the authors, this field is not standardized by any implementation.

AFSVOL_TLV_TAG_VOL_QUOTA_RESERVATION

This tuple MUST have payload of type AFSVOL_TLV_TYPE_DISK_BLOCKS. This field, otherwise known as minquota, specifies the amount of storage (in units of 1024 octets) that are reserved on the underlying storage for use by this volume.

AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY_DATE

This tuple MUST have payload of type AFSVOL_TLV_TYPE_TIME_ABS. This field, otherwise known as dayUseDate, specifies the timestamp when AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY was reset to zero, and the previous value rolled over to index 0 of AFSVOL_TLV_TAG_VOL_STAT_USE_PER_DOW.

6.4. Day-of-week usage statistics

The day-of-week usage statistics accessed via tag AFSVOL_TLV_TAG_VOL_STAT_USE_PER_DOW provide access to historic data for the 7 days prior to the current access counter available via tag AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY. Depending on the desired mode of statistics collection, two qualifier types are supported by this tag.

6.4.1. Qualifiers

6.4.1.1. NULL qualifier

When the qualifier is of type AFSVOL_TLV_TYPE_NULL, then a custom payload of type AFSVOL_TLV_TYPE_VOL_DOW_USE will be used to deliver day-of-week usage data for the past week. This type is defined as follows:

```
struct AFSVol_stat_use_per_dow {
    afs_uint64 stat_dow[7];
    afs_uint32 stat_flags;
};
```

Figure 6

Seven bits in the stat_flags field are used to assert data validity for each day of week. These bits are present to help monitoring applications distinguish between days for which no data was collected (e.g. due to the volume being less than eight days old) and days when there were exactly zero accesses. These bits are defined as follows:

Flag	Description
AFSVOL_VOL_STAT_DOW0_VALID	stat_dow[0] is valid
AFSVOL_VOL_STAT_DOW1_VALID	stat_dow[1] is valid
AFSVOL_VOL_STAT_DOW2_VALID	stat_dow[2] is valid
AFSVOL_VOL_STAT_DOW3_VALID	stat_dow[3] is valid
AFSVOL_VOL_STAT_DOW4_VALID	stat_dow[4] is valid
AFSVOL_VOL_STAT_DOW5_VALID	stat_dow[5] is valid
AFSVOL_VOL_STAT_DOW6_VALID	stat_dow[6] is valid
AFSVOL_VOL_STAT_DOW_FUZZY	server incapable of guaranteeing validity

Day-of-week statistics flags

Server implementations which are incapable of distinguishing between days when there was no usage, and for which there is no data SHOULD make a best-effort to populate the 7 per-day bits, and MUST assert the 0x80 stat_flags bit.

6.4.1.2. UINT64 qualifier

When the qualifier is of type AFSVOL_TLV_TYPE_UINT64, then a payload of type AFSVOL_TLV_TYPE_UINT64 will be used to deliver day-of-week usage data for the day of week specified in the uint64 qualifier. Valid qualifiers are in the range 0 to 6, where 0 means the day prior to the current day, and 6 means 7 days prior to the current day.

6.4.2. Calendar day correlation

Clients who need to poll AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY or AFSVOL_TLV_TAG_VOL_STAT_USE_PER_DOW, and need to correlate this statistical data with specific calendar days SHOULD simultaneously query for the value stored at tag AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY_DATE. By querying these tags in the same RPC invocation, the caller will be able correlate the usage statistics with calendar days in a race-free manner. Querying AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY_DATE in a separate RPC invocation is not guarnteed to yield correct results, as there is no way to guarantee the value didn't change between the two RPC invocations.

7. Extended volume state exportation

In addition to exporting the existing volser state, DAFS state metadata will also be exported via the TLV interface. Specifically, an extended volume state field, and a raw DAFS state debugging tag, will be exported.

7.1. Volume state explanations

Given that volume state information is useful across all server implementations, a collection of generic state explanations shall be standardized. These standardized enumeration values shall be published via a special volume state explanation tag. The following states are initially defined in the namespace:

AFSVOL_VOL_STATE_EXPL_NONE

No further explanation is deemed necessary.

AFSVOL_VOL_STATE_EXPL_UNKNOWN

This volume is in its current state for unknown reasons.

AFSVOL_VOL_STATE_EXPL_OUT_OF_SERVICE

This volume is administratively out of service. For example, the IBM AFS and OpenAFS implementations both permit an administrator to force a volume offline by mutating the blessed or inService disk header bits.

AFSVOL_VOL_STATE_EXPL_DELETED

This volume no longer exists on-disk. This record merely serves as a pointer to tell clients that the volume has been permanently deleted, or moved to a new location.

AFSVOL_VOL_STATE_EXPL_READY

This volume is ready to service requests. If the primary volume state is offline, this means the volume is ready to be brought online as soon as a remote procedure call needs to access this volume.

AFSVOL_VOL_STATE_EXPL_ATTACHING

This volume is busy attaching. Assuming the process completes successfully, the volume will be brought online.

AFSVOL_VOL_STATE_EXPL_DETACHING

This volume is busy detaching.

AFSVOL_VOL_STATE_EXPL_BUSY

This volume is busy performing some ancillary operation which requires exclusive access.

AFSVOL_VOL_STATE_EXPL_IO_BUSY

This volume is busy performing an I/O operation which requires exclusive access.

AFSVOL_VOL_STATE_EXPL_SALVAGING

This volume is currently being salvaged in the background.

AFSVOL_VOL_STATE_EXPL_SALVAGE_NEEDED

This volume is offline, and will require a salvage before it can be brought online.

AFSVOL_VOL_STATE_EXPL_ERROR

This volume has been forced offline due to a non-recoverable error. Manual intervention by an administrator will be necessary to bring this volume back to an operable state.

AFSVOL_VOL_STATE_EXPL_VOLUME_OPERATION

This volume is currently offline because a volume transaction requires exclusive access.

```
enum AFSVol_vol_state_expl {
    AFSVOL_VOL_STATE_EXPL_NONE = 0,
    AFSVOL_VOL_STATE_EXPL_UNKNOWN = 1,
    AFSVOL_VOL_STATE_EXPL_OUT_OF_SERVICE = 2,
    AFSVOL_VOL_STATE_EXPL_DELETED = 3,
    AFSVOL_VOL_STATE_EXPL_READY = 4,
    AFSVOL_VOL_STATE_EXPL_ATTACHING = 5,
    AFSVOL_VOL_STATE_EXPL_DETACHING = 6,
    AFSVOL_VOL_STATE_EXPL_BUSY = 7,
    AFSVOL_VOL_STATE_EXPL_IO_BUSY = 8,
    AFSVOL_VOL_STATE_EXPL_SALVAGING = 9,
    AFSVOL_VOL_STATE_EXPL_SALVAGE_NEEDED = 10,
    AFSVOL_VOL_STATE_EXPL_ERROR = 11,
    AFSVOL_VOL_STATE_EXPL_VOLUME_OPERATION = 12
};
```

XDR definition of Volume State Enumeration

7.2. Mapped process types

It is useful to be able to track volume ownership by process type. In order to do this, a new program type namespace must be defined. The following types are initially defined in the program type namespace:

AFSVOL_PROGRAM_TYPE_NONE

This value refers to the absence of a process.

AFSVOL_PROGRAM_TYPE_FILE_SERVER

An afs file server process (Rx service ID 1).

AFSVOL_PROGRAM_TYPE_VOLUME_SERVER

An afs volume server process (Rx service ID 4).

AFSVOL_PROGRAM_TYPE_SALVAGER

An afs stand-alone salvager process.

AFSVOL_PROGRAM_TYPE_SALVAGE_SERVER

An OpenAFS DAFS salvage server process.

AFSVOL_PROGRAM_TYPE_VOLUME_UTILITY

Any ancillary stand-alone volume utility process.

AFSVOL_PROGRAM_TYPE_UNKNOWN

This value refers to an unknown process type.

```
enum AFSVol_program_type {
    AFSVOL_PROGRAM_TYPE_NONE = 0,
    AFSVOL_PROGRAM_TYPE_FILE_SERVER = 1,
    AFSVOL_PROGRAM_TYPE_VOLUME_SERVER = 2,
    AFSVOL_PROGRAM_TYPE_SALVAGER = 3,
    AFSVOL_PROGRAM_TYPE_SALVAGE_SERVER = 4,
    AFSVOL_PROGRAM_TYPE_VOLUME_UTILITY = 5,
    AFSVOL_PROGRAM_TYPE_UNKNOWN = 6
};
```

XDR definition of Program Type Enumeration

7.3. TLV tuples

Volume state will be exported via five new TLV tuples:

AFSVOL_TLV_TAG_VOL_STATE_ONLINE

This tuple MUST have payload of either type AFSVOL_TLV_TYPE_TRUE, or AFSVOL_TLV_TYPE_FALSE. This value SHALL tell the caller whether or not the volume is fully online.

AFSVOL_TLV_TAG_VOL_STATE_AVAILABLE

This tuple MUST have payload of either type AFSVOL_TLV_TYPE_TRUE, or AFSVOL_TLV_TYPE_FALSE. This tuple shall tell the caller whether or not the volume is available. This SHOULD be asserted

either when the volume is fully online, or when the volume can be brought online on-demand within a reasonable length of time following receipt of an RPC call to Rx service id 1 requesting access to the volume.

AFSVOL_TLV_TAG_VOL_STATE_EXPL

This tuple MUST have payload of type AFSVOL_TLV_TYPE_UINT64. The u_u64 payload shall contain a volume state explanation enumeration value, as defined in Section 7.1.

AFSVOL_TLV_TAG_VOL_STATE_DAFS_RAW

For servers exporting capability AFSVOL_CAPABILITY_DAFS, this payload MUST be of type AFSVOL_TLV_TYPE_OPAQUE. Encoding of raw state is unspecified and implementation-private.

AFSVOL_TLV_TAG_VOL_STATE_OWNING_PROCESS

This tag should only be advertised as available on server implementations which support tracking volume ownership by process type. When available, this payload MUST be of type AFSVOL_TLV_TYPE_UINT64. The u_u64 payload shall contain a program type enumeration value, as defined in Section 7.2.

8. AFS-3 Object Storage Extensions Policy Attributes

RxOSD [AFS-OSD08] [AFS-OSD09] requires two TLV tuples to encode new quota types:

AFSVOL_TLV_TAG_VOL_QUOTA_BLOCKS_STORED_LOCALLY

The value in this tuple defines the maximum allowable storage, in units of blocks, that may be stored on the local file server partition. When storage is required beyond this limit, some data must be migrated to object storage devices (OSDs). This tuple MUST have a payload of type AFSVOL_TLV_TYPE_DISK_BLOCKS.

AFSVOL_TLV_TAG_VOL_QUOTA_FILES

The value in this tuple defines the maximum allowable file count for this volume. This tuple MUST have a payload of type AFSVOL_TLV_TYPE_UINT64.

9. Backward Compatibility

AFSVol services providing extended Tag-Length-Value RPCs MUST provide backwards compatible interfaces to both legacy clients and servers. Additionally, interoperability between TLV versions must also be specified if they do not comply with the following requirements:

1. AFSVol TLV servers replying to legacy AFSVol clients MUST provide the identical response to an AFSVol server.

2. AFSVol TLV clients communicating with AFSVol servers MUST fall back to using non-TLV AFSVol RPCs.

3. AFSVol TLV clients to AFSVol TLV servers:

   A. Where capabilities match or the server can provide capabilities including those which the client requests, the server MUST reply with exactly the capabilities requested.

   B. Where the client requests capabilities that the server does not provide it MUST either return an 'unknown tag' error code, or (OPTIONAL) fall back to an non-TLV AFSVol response.

10. Acknowledgements

We would like to thank all of the participants at the 2009 Edinburgh AFS hackathon for their input into the design of this TLV mechanism. Alistair Ferguson has provided much useful feedback, especially with regard to backwards compatibility and discriminated union type identifier namespace allocations. Andrew Deason and Michael Meffie have provided considerable input with regard to the discriminated union XDR decoding problem, AFS registrar and namespace allocation concerns, what metadata should be exported in the initial revision, the notion of data qualifiers, as well as commentary about how they envision this extension being used to support future protocol extensions. Derrick Brashear has provided helpful feedback with regard to restructuring the volume state reporting tags. Thanks to Christof Hanke and Hartmut Reuter for collaborating to make this memo compatible with their RxOSD protocol enhancments, and, furthermore, for providing helpful feedback regarding the language in this draft. Finally, special thanks to Jeffrey Hutzelman for providing considerable help with restructuring this memo to improve readability and limit its scope to something tractable.

11. IANA Considerations

This memo includes no request to IANA.

12. AFS Assign Numbers Registrar Considerations

The AFS Assigned Numbers Registrar will need to consider several assigned numbers requests.

12.1. Namespace allocations

First and foremost, this memo requests that the AFS Registrar assume control over several new registries:

1. AFSVol TLV payload type namespace

2. AFSVol TLV tag namespace

3. AFSVol TLV flag namespace

4. AFSVol TLV Day-of-Week Stats flag namespace

5. AFSVol Mapped Volume State namespace

6. AFSVol Program Type namespace

12.1.1. AFSVol TLV Payloads

This memo requests the allocation of a new registry with the formal name "AFSVol TLV Payloads". This registry will be used to track allocations of enumeration values in the AFSVol_TLV_type XDR enum, and the mapping of these values onto their respective XDR type definitions. This is a 32-bit unsigned namespace. Allocations can fall into one of a few categories:

Range	Description
0 to 0xfeffffff	- AFS-STDS Early Assignment
0xff000000 to 0xfffeffff	- Private Assignment
0xffff0000 to 0xffffffff	- reserved

Subdivision into allocation policy regions

In the table above, "AFS-STDS Early Assignment" refers to the

allocation policy described in [I-D.wilkinson-afs3-standardisation]; "Private Assignment", and "Reserved" are as-described in [RFC5226].

Allocation requests for the "AFS-STDS Early Assignment" region MUST contain the following information:

- type name
- RFC section reference to definition of data encoding associated with this type enumeration value

In addition, an "AFS-STDS Early Assignment" allocation request MAY include the following optional elements:

- type description
- desired value in AFSVol_TLV_type enumeration
- RFC section reference to discussion regarding backwards compatibility
- RFC section reference to relevant security considerations

12.1.2. AFSVol TLV Tags

This memo requests the allocation of a new registry with the formal name "AFSVol TLV Tags". This registry will be used to track allocations of enumeration values in the AFSVol_TLV_tag XDR enum, and the mapping of these values onto legal tags and qualifiers. This is a 32-bit unsigned namespace. Allocations can fall into one of a few categories:

Range	Description
0 to 0xfeffffff	- AFS-STDS Early Assignment
0xff000000 to 0xfffeffff	- Private Assignment
0xffff0000 to 0xffffffff	- reserved

Subdivision into allocation policy regions

In the table above, "AFS-STDS Early Assignment" refers to the allocation policy described in [I-D.wilkinson-afs3-standardisation]; "Private Assignment", and "Reserved" are as-described in [RFC5226].

Allocation requests for the "AFS-STDS Early Assignment" region MUST contain the following information:

- tag name
- RFC section reference to definition of tag semantics

In addition, an "AFS-STDS Early Assignment" allocation request MAY include the following optional elements:

- tag description
- desired value in AFSVol_TLV_tag enumeration
- RFC section reference to definition of qualifier semantics for this tag
- RFC section reference to discussion regarding backwards compatibility
- RFC section reference to relevant security considerations

12.1.3. AFSVol TLV Flags

This memo requests the allocation of a new registry with the formal name "AFSVol TLV Flags". This registry will be used to track allocations of flag bits in the AFSVol_TLV.tlv_flags field. This is a 32-bit flag namespace. All flag bit allocations shall fall under the "AFS-STDS Early Assignment" allocation policy, as described in [I-D.wilkinson-afs3-standardisation]. Flag bit allocation requests MUST contain the following information:

- flag name
- RFC section reference to definition of flag semantics

In addition, an allocation request MAY include the following optional elements:

- flag description
- desired flag bit value
- RFC section reference to discussion regarding backwards compatibility
- RFC section reference to relevant security considerations

12.1.4. AFSVol DoW Stats Flags

This memo requests the allocation of a new registry with the formal name "AFSVol DoW Stats Flags". This registry will be used to track allocations of flag bits in the AFSVol_stat_use_per_dow.stat_flags field. This is a 32-bit flag namespace. All flag bit allocations shall fall under the "AFS-STDS Early Assignment" allocation policy, as described in [I-D.wilkinson-afs3-standardisation]. Flag bit allocation requests MUST contain the following information:

- flag name
- RFC section reference to definition of flag semantics

In addition, an allocation request MAY include the following optional elements:

- flag description
- desired flag bit value
- RFC section reference to discussion regarding backwards compatibility
- RFC section reference to relevant security considerations

12.1.5. AFSVol Vol State Expls

This memo requests the allocation of a new registry with the formal name "AFSVol Vol State Expls". This registry will be used to track allocations of enumeration values in the AFSVol_vol_state_expl enum (see Section 7.1). This is a 32-bit unsigned namespace. Allocations can fall into one of a few categories:

Range	Description
0 to 0xfeffffff	- AFS-STDS Early Assignment
0xff000000 to 0xffffffff	- Private Assignment

Subdivision into allocation policy regions

In the table above, "AFS-STDS Early Assignment" refers to the allocation policy described in [I-D.wilkinson-afs3-standardisation]; "Private Assignment" is as-described in [RFC5226].

Allocation requests for the "AFS-STDS Early Assignment" region MUST

contain the following information:

- state name
- RFC section reference to definition of this volume state enumeration value

In addition, an "AFS-STDS Early Assignment" allocation request MAY include the following optional elements:

- state description
- desired value in AFSVol_vol_state_expl enumeration
- RFC section reference to discussion regarding backwards compatibility
- RFC section reference to relevant security considerations

12.1.6. AFSVol Program Types

This memo requests the allocation of a new registry with the formal name "AFSVol Program Types". This registry will be used to track allocations of enumeration values in the AFSVol_program_type enum (see Section 7.2). This is a 32-bit unsigned namespace. Allocations can fall into one of a few categories:

Range	Description
0 to 0xefffffff	- AFS-STDS Early Assignment
0xff000000 to 0xffffffff	- Private Assignment

Subdivision into allocation policy regions

In the table above, "AFS-STDS Early Assignment" refers to the allocation policy described in [I-D.wilkinson-afs3-standardisation]; "Private Assignment" is as-described in [RFC5226].

Allocation requests for the "AFS-STDS Early Assignment" region MUST contain the following information:

- program name
- RFC section reference to definition of this program type enumeration value

In addition, an "AFS-STDS Early Assignment" allocation request MAY include the following optional elements:

- program description
- desired value in AFSVol_program_type enumeration
- RFC section reference to discussion regarding backwards compatibility
- RFC section reference to relevant security considerations

12.2. Assigned numbers allocations

In addition to requesting the allocation of new registries, this memo also requests several new allocations within existing assigned numbers registries.

12.2.1. VICED Capability bits

One new capability bit is requested:

- VICED_CAPABILITY_DAFS (see Section 3)

12.2.2. AFSVol Capabilities

The following allocations are requested in the "AFSVol Capabilites" registry [I-D.keiser-afs3-capabilities]:

- AFSVOL_CAPABILITY_DAFS = 0x1 (see Section 3)
- AFSVOL_CAPABILITY_TLV = 0x2 (see Section 3)

12.2.3. AFSVol TLV Payloads

The following initial allocations are requested in the newly-created registry "AFSVol TLV Payloads":

- AFSVOL_TLV_TYPE_NULL = 0 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_TRUE = 1 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_FALSE = 2 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_UINT64 = 3 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_UINT64_VEC = 4 (see Section 4.1.1)

- AFSVOL_TLV_TYPE_INT64 = 5 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_INT64_VEC = 6 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_UUID = 7 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_STRING = 8 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_TIME_ABS = 9 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_TIME_ABS_VEC = 10 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_TIME_REL = 11 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_TIME_REL_VEC = 12 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_VOL_ID = 13 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_VOL_ID_VEC = 14 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_PART_ID = 15 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_PART_ID_VEC = 16 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_DISK_BLOCKS = 17 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_STAT_COUNTER = 18 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_STAT_GAUGE = 19 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_BIT64 = 20 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_VOL_DOW_USE = 21 (see Section 4.1.1)
- AFSVOL_TLV_TYPE_OPAQUE = 22 (see Section 4.1.1)

12.2.4. AFSVol TLV Tags

The following initial allocations are requested in the newly-created registry "AFSVol TLV Tags":

- AFSVOL_TLV_TAG_EOS = 0 (see Section 5.4.1)
- AFSVOL_TLV_TAG_VOL_NAME = 1 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_STATUS = 2 (see Section 6.1)

- AFSVOL_TLV_TAG_VOL_IN_USE = 3 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_ID = 4 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_TYPE = 5 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_CLONE_ID = 6 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_BACKUP_ID = 7 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_PARENT_ID = 8 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_COPY_DATE = 9 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_CREATE_DATE = 10 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_ACCESS_DATE = 11 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_UPDATE_DATE = 12 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_BACKUP_DATE = 13 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_SIZE = 14 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_FILE_COUNT = 15 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_QUOTA_BLOCKS = 16 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY = 17 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_STAT_USE_PER_DOW = 18 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_STAT_READS = 19 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_STAT_WRITES = 20 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_STAT_FILE_SAME_AUTHOR = 21 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_STAT_FILE_DIFFERENT_AUTHOR = 22 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_STAT_DIR_SAME_AUTHOR = 23 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_STAT_DIR_DIFFERENT_AUTHOR = 24 (see Section 6.1)
- AFSVOL_TLV_TAG_VOL_TRANS_ID = 25 (see Section 6.2)

- AFSVOL_TLV_TAG_VOL_TRANS_TIME = 26 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_TRANS_CREATE_TIME = 27 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_TRANS_RETURN_CODE = 28 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_TRANS_ATTACH_MODE = 29 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_TRANS_STATUS = 30 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_TRANS_FLAGS = 31 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_TRANS_LAST_PROC_NAME = 32 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_TRANS_CALL_VALID = 33 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_TRANS_READ_NEXT = 34 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_TRANS_XMIT_NEXT = 35 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_TRANS_LAST_RECV_TIME = 36 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_TRANS_LAST_SEND_TIME = 37 (see Section 6.2)
- AFSVOL_TLV_TAG_VOL_IN_SERVICE = 38 (see Section 6.3)
- AFSVOL_TLV_TAG_VOL_BLESSED = 39 (see Section 6.3)
- AFSVOL_TLV_TAG_VOL_RESTORED_FROM_ID = 40 (see Section 6.3)
- AFSVOL_TLV_TAG_VOL_DESTROYED = 41 (see Section 6.3)
- AFSVOL_TLV_TAG_VOL_NEEDS_SALVAGE = 42 (see Section 6.3)
- AFSVOL_TLV_TAG_VOL_OFFLINE_MESSAGE = 43 (see Section 6.3)
- AFSVOL_TLV_TAG_VOL_EXPIRATION_DATE = 44 (see Section 6.3)
- AFSVOL_TLV_TAG_VOL_QUOTA_RESERVATION = 45 (see Section 6.3)
- AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY_DATE = 46 (see Section 6.3)
- AFSVOL_TLV_TAG_VOL_STATE_ONLINE = 47 (see Section 7)
- AFSVOL_TLV_TAG_VOL_STATE_AVAILABLE = 48 (see Section 7)
- AFSVOL_TLV_TAG_VOL_STATE_EXPL = 49 (see Section 7)

- AFSVOL_TLV_TAG_VOL_STATE_DAFS_RAW = 50 (see Section 7)
- AFSVOL_TLV_TAG_VOL_STATE_OWNING_PROCESS = 51 (see Section 7)
- AFSVOL_TLV_TAG_VOL_QUOTA_BLOCKS_STORED_LOCALLY = 52 (see Section 8)
- AFSVOL_TLV_TAG_VOL_QUOTA_FILES = 53 (see Section 8)

12.2.5. AFSVol TLV Flags

The following initial allocations are requested within the newly-created registry "AFSVol TLV Flags":

- AFSVOL_TLV_FLAG_UNSUPPORTED = 0x1 (see Section 4.1.2)
- AFSVOL_TLV_FLAG_READ_ERROR = 0x2 (see Section 4.1.2)
- AFSVOL_TLV_FLAG_CRITICAL = 0x4 (see Section 4.1.2)
- AFSVOL_TLV_FLAG_QUALIFIER_NO_MATCH = 0x8 (see Section 4.1.2)
- AFSVOL_TLV_FLAG_MORE = 0x10 (see Section 4.1.2)
- AFSVOL_TLV_FLAG_OBJ_NOT_SUPP = 0x20 (see Section 4.1.2)

12.2.6. AFSVol DoW Stats Flags

The following initial allocations are requested within the newly-created registry "AFSVol DoW Stats Flags":

- AFSVOL_VOL_STAT_DOW0_VALID = 0x1 (see Section 6.4)
- AFSVOL_VOL_STAT_DOW1_VALID = 0x2 (see Section 6.4)
- AFSVOL_VOL_STAT_DOW2_VALID = 0x4 (see Section 6.4)
- AFSVOL_VOL_STAT_DOW3_VALID = 0x8 (see Section 6.4)
- AFSVOL_VOL_STAT_DOW4_VALID = 0x10 (see Section 6.4)
- AFSVOL_VOL_STAT_DOW5_VALID = 0x20 (see Section 6.4)
- AFSVOL_VOL_STAT_DOW6_VALID = 0x40 (see Section 6.4)
- AFSVOL_VOL_STAT_DOW_FUZZY = 0x80 (see Section 6.4)

12.2.7. VOLS Error Table

Within the VOLS error table (offset 1492325120), several new codes need to be allocated:

- VOLSER_TAG_UNSUPPORTED
- VOLSER_TAG_READ_ONLY
- VOLSER_TAG_WRITE_FAILED
- VOLSER_TAG_DECODE_FAILED
- VOLSER_TAG_UNSUPPORTED_ENCODING
- VOLSER_TLV_QUALIFIER_UNSUPPORTED_ENCODING
- VOLSER_TLV_QUALIFIER_DECODE_FAILED
- VOLSER_TLV_QUALIFIER_INVALID
- VOLSER_TRANS_INVALID

12.2.8. AFSVol Vol State Expls

The following initial allocations are requested within the newly-created registry "AFSVol Vol State Expls":

- AFSVOL_VOL_STATE_EXPL_NONE = 0 (see Section 7.1)
- AFSVOL_VOL_STATE_EXPL_UNKNOWN = 1 (see Section 7.1)
- AFSVOL_VOL_STATE_EXPL_OUT_OF_SERVICE = 2 (see Section 7.1)
- AFSVOL_VOL_STATE_EXPL_DELETED = 3 (see Section 7.1)
- AFSVOL_VOL_STATE_EXPL_READY = 4 (see Section 7.1)
- AFSVOL_VOL_STATE_EXPL_ATTACHING = 5 (see Section 7.1)
- AFSVOL_VOL_STATE_EXPL_DETACHING = 6 (see Section 7.1)
- AFSVOL_VOL_STATE_EXPL_BUSY = 7 (see Section 7.1)
- AFSVOL_VOL_STATE_EXPL_IO_BUSY = 8 (see Section 7.1)
- AFSVOL_VOL_STATE_EXPL_SALVAGING = 9 (see Section 7.1)

- AFSVOL_VOL_STATE_EXPL_SALVAGE_NEEDED = 10 (see Section 7.1)
- AFSVOL_VOL_STATE_EXPL_ERROR = 11 (see Section 7.1)
- AFSVOL_VOL_STATE_EXPL_VOLUME_OPERATION = 12 (see Section 7.1)

12.2.9. AFSVol Program Types

Within the new AFS program type namespace, the following allocations are requested:

- AFSVOL_PROGRAM_TYPE_NONE = 0 (see Section 7.2)
- AFSVOL_PROGRAM_TYPE_FILE_SERVER = 1 (see Section 7.2)
- AFSVOL_PROGRAM_TYPE_VOLUME_SERVER = 2 (see Section 7.2)
- AFSVOL_PROGRAM_TYPE_SALVAGER = 3 (see Section 7.2)
- AFSVOL_PROGRAM_TYPE_SALVAGE_SERVER = 4 (see Section 7.2)
- AFSVOL_PROGRAM_TYPE_VOLUME_UTILITY = 5 (see Section 7.2)
- AFSVOL_PROGRAM_TYPE_UNKNOWN = 6 (see Section 7.2)

13. Security Considerations

Security and authorization issues are tag-specific. Most known implementations of the legacy AFSVol RPCs permitted rxnull connections to perform the four ListVolume RPCs, and AFSVolMonitor. Arguably, it is time to re-evaluate this decision, and subsequently restrict access to some tags, as they do permit potentially sensitive volume--or operational--metadata to leak onto public networks.

14. References

14.1. Normative References

[I-D.deason-afs3-type-time]
Deason, A., "Base Types for Time in AFS-3", draft-deason-afs3-type-time-03 (work in progress), August 2011.

[I-D.keiser-afs3-capabilities]
Keiser, T., Jenkins, S., and A. Deason, "AFS-3 Protocol Capabilities Query Mechanism",

draft-keiser-afs3-capabilities-00 (work in progress), September 2012.

[I-D.keiser-afs3-xdr-primitive-types]
Keiser, T. and A. Deason, "AFS-3 Rx RPC XDR Primitive Type Definitions", draft-keiser-afs3-xdr-primitive-types-01 (work in progress), September 2012.

[I-D.keiser-afs3-xdr-union]
Keiser, T. and A. Deason, "Extensible XDR Discriminated Union Primitive Type", draft-keiser-afs3-xdr-union-06 (work in progress), September 2012.

[I-D.wilkinson-afs3-standardisation]
Wilkinson, S., "Options for AFS Standardisation", draft-wilkinson-afs3-standardisation-00 (work in progress), June 2010.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

14.2. Informative References

[AFS-OSD08]
Tobbicke, R., Maslennikov, A., Giammarino, L., Belloni, R., and H. Reuter, "AFS + Object Storage", AFS and Kerberos Best Practices Workshop 2008, May 2008, <http://workshop.openafs.org/afsbpw08/talks/thu_3/OpenAFS+ObjectStorage.pdf>.

[AFS-OSD09]
Reuter, H., Frank, F., and A. Maslennikov, "Embedded Filesystems (Direct Client Access to Vice Partitions)", AFS and Kerberos Best Practices Workshop 2009, June 2009, <http://workshop.openafs.org/afsbpw09/talks/thu_2/Embedded_filesystems_opt.pdf>.

[AFS3-RX] Zayas, E., "AFS-3 Programmer's Reference: Specification for the Rx Remote Procedure Call Facility", Transarc Corp. Tech. Rep. FS-00-D164, August 1991.

[AFS3-VVL]
Zayas, E., "AFS-3 Programmer's Reference: Volume Server/ Volume Location Server Interface", Transarc Corp. Tech.

Rep. FS-00-D165, August 1991.

[CMU-ITC-83-025]
Morris, J., Van Houweling, D., and K. Slack, "The Information Technology Center", CMU ITC Tech. Rep. CMU-ITC-83-025, 1983.

[CMU-ITC-84-020]
West, M., "VICE File System Services", CMU ITC Tech. Rep. CMU-ITC-84-020, August 1984.

[CMU-ITC-85-039]
Satyanarayanan, M., Howard, J., Nichols, D., Sidebotham, R., Spector, A., and M. West, "The ITC Distributed File System: Principles and Design", Proc. 10th ACM Symp. Operating Sys. Princ. Vol. 19, No. 5, December 1985.

[CMU-ITC-87-068]
Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., and M. West, "Scale and Performance in a Distributed File System", ACM Trans. Comp. Sys. Vol. 6, No. 1, pp. 51-81, February 1988.

[CMU-ITC-88-062]
Howard, J., "An Overview of the Andrew File System"", Proc. 1988 USENIX Winter Tech. Conf. pp. 23-26, February 1988.

[CMU-ITC-88-070]
Zayas, E. and C. Everhart, "Design and Specification of the Cellular Andrew Environment", CMU ITC Tech. Rep. CMU-ITC-88-070, August 1988.

[DAFS] Keiser, T., "Demand Attach / Fast-restart File Server", AFS and Kerberos Best Practices Workshop 2006, June 2006, <http://workshop.openafs.org/afsbpw06/talks/tkeiser-dafs.pdf>.

[RFC4506] Eisler, M., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, May 2009.

Appendix A. Sample Rx RPC-L Definition for AFSVol TLV Mechanism

```
const AFSVOL_TLV_TAG_MAX = 1024;         /* upper-bound on number of
```

```
                                          * TLV tuples per RPC */
   const AFSVOL_TLV_OPAQUE_MAX = 262144;    /* upper-bound on size of
                                          * value payload */
   const AFSVOL_TLV_UINT64_MAX = 32768;     /* upper-bound on length of
                                             uint64 vector payload */
   const AFSVOL_TLV_TIME_MAX = 21845;       /* upper-bound on length of
                                             AFSTime vector payload */
   const AFSVOL_BULK_GETVOLUME_MAX = 1024;  /* upper-bound on
                                          * (partition, volume)
                                          * tuples per RPC */

   const AFSVOL_TLV_FLAG_UNSUPPORTED = 0x1;
   const AFSVOL_TLV_FLAG_READ_ERROR = 0x2;
   const AFSVOL_TLV_FLAG_CRITICAL = 0x4;
   const AFSVOL_TLV_FLAG_QUALIFIER_NO_MATCH = 0x8;
   const AFSVOL_TLV_FLAG_MORE = 0x10;
   const AFSVOL_TLV_FLAG_OBJ_NOT_SUPP = 0x20;

   enum AFSVol_TLV_type {
       AFSVOL_TLV_TYPE_NULL          = 0,
       AFSVOL_TLV_TYPE_TRUE          = 1,
       AFSVOL_TLV_TYPE_FALSE         = 2,
       AFSVOL_TLV_TYPE_UINT64        = 3,
       AFSVOL_TLV_TYPE_UINT64_VEC    = 4,
       AFSVOL_TLV_TYPE_INT64         = 5,
       AFSVOL_TLV_TYPE_INT64_VEC     = 6,
       AFSVOL_TLV_TYPE_UUID          = 7,
       AFSVOL_TLV_TYPE_STRING        = 8,
       AFSVOL_TLV_TYPE_TIME_ABS      = 9,
       AFSVOL_TLV_TYPE_TIME_ABS_VEC  = 10,
       AFSVOL_TLV_TYPE_TIME_REL      = 11,
       AFSVOL_TLV_TYPE_TIME_REL_VEC  = 12,
       AFSVOL_TLV_TYPE_VOL_ID        = 13,
       AFSVOL_TLV_TYPE_VOL_ID_VEC    = 14,
       AFSVOL_TLV_TYPE_PART_ID       = 15,
       AFSVOL_TLV_TYPE_PART_ID_VEC   = 16,
       AFSVOL_TLV_TYPE_DISK_BLOCKS   = 17,
       AFSVOL_TLV_TYPE_STAT_COUNTER  = 18,
       AFSVOL_TLV_TYPE_STAT_GAUGE    = 19,
       AFSVOL_TLV_TYPE_BIT64         = 20,
       AFSVOL_TLV_TYPE_VOL_DOW_USE   = 21,
       AFSVOL_TLV_TYPE_OPAQUE        = 22
   };

   const AFSVOL_VOL_STAT_DOW0_VALID = 0x1;
   const AFSVOL_VOL_STAT_DOW1_VALID = 0x2;
   const AFSVOL_VOL_STAT_DOW2_VALID = 0x4;
   const AFSVOL_VOL_STAT_DOW3_VALID = 0x8;
```

```
   const AFSVOL_VOL_STAT_DOW4_VALID = 0x10;
   const AFSVOL_VOL_STAT_DOW5_VALID = 0x20;
   const AFSVOL_VOL_STAT_DOW6_VALID = 0x40;
   const AFSVOL_VOL_STAT_DOW_FUZZY  = 0x80;

   struct AFSVol_stat_use_per_dow {
       afs_uint64 stat_dow[7];
       afs_uint32 stat_flags;
   };

   enum AFSVol_vol_state_expl {
       AFSVOL_VOL_STATE_EXPL_NONE = 0,
       AFSVOL_VOL_STATE_EXPL_UNKNOWN = 1,
       AFSVOL_VOL_STATE_EXPL_OUT_OF_SERVICE = 2,
       AFSVOL_VOL_STATE_EXPL_DELETED = 3,
       AFSVOL_VOL_STATE_EXPL_READY = 4,
       AFSVOL_VOL_STATE_EXPL_ATTACHING = 5,
       AFSVOL_VOL_STATE_EXPL_DETACHING = 6,
       AFSVOL_VOL_STATE_EXPL_BUSY = 7,
       AFSVOL_VOL_STATE_EXPL_IO_BUSY = 8,
       AFSVOL_VOL_STATE_EXPL_SALVAGING = 9,
       AFSVOL_VOL_STATE_EXPL_SALVAGE_NEEDED = 10,
       AFSVOL_VOL_STATE_EXPL_ERROR = 11,
       AFSVOL_VOL_STATE_EXPL_VOLUME_OPERATION = 12
   };

   enum AFSVol_program_type {
       AFSVOL_PROGRAM_TYPE_NONE = 0,
       AFSVOL_PROGRAM_TYPE_FILE_SERVER = 1,
       AFSVOL_PROGRAM_TYPE_VOLUME_SERVER = 2,
       AFSVOL_PROGRAM_TYPE_SALVAGER = 3,
       AFSVOL_PROGRAM_TYPE_SALVAGE_SERVER = 4,
       AFSVOL_PROGRAM_TYPE_VOLUME_UTILITY = 5,
       AFSVOL_PROGRAM_TYPE_UNKNOWN = 6
   };

   ext-union AFSVol_TLV_value switch(AFSVol_TLV_type type) {
    case AFSVOL_TLV_TYPE_NULL:
       void;

    case AFSVOL_TLV_TYPE_TRUE:
       void;

    case AFSVOL_TLV_TYPE_FALSE:
       void;

    case AFSVOL_TLV_TYPE_UINT64:
       afs_uint64 u_u64;
```

```
    case AFSVOL_TLV_TYPE_VOL_ID:
       afs_uint64 u_vol_id;

    case AFSVOL_TLV_TYPE_PART_ID:
       afs_uint64 u_part_id;

    case AFSVOL_TLV_TYPE_DISK_BLOCKS:
       afs_uint64 u_disk_blocks;

    case AFSVOL_TLV_TYPE_STAT_COUNTER:
       afs_uint64 u_stat_counter;

    case AFSVOL_TLV_TYPE_BIT64:
       afs_uint64 u_bit64;

    case AFSVOL_TLV_TYPE_INT64:
       afs_int64 u_s64;

    case AFSVOL_TLV_TYPE_STAT_GAUGE:
       afs_int64 u_stat_gauge;

    case AFSVOL_TLV_TYPE_UINT64_VEC:
       afs_uint64 u_u64_vec<AFSVOL_TLV_UINT64_MAX>;

    case AFSVOL_TLV_TYPE_VOL_ID_VEC:
       afs_uint64 u_vol_id_vec<AFSVOL_TLV_UINT64_MAX>;

    case AFSVOL_TLV_TYPE_PART_ID_VEC:
       afs_uint64 u_part_id_vec<AFSVOL_TLV_UINT64_MAX>;

    case AFSVOL_TLV_TYPE_INT64_VEC:
       afs_int64 u_s64_vec<AFSVOL_TLV_UINT64_MAX>;

    case AFSVOL_TLV_TYPE_TIME_ABS:
       AFSTime u_time_abs;

    case AFSVOL_TLV_TYPE_TIME_REL:
       AFSRelTimestamp u_time_rel;

    case AFSVOL_TLV_TYPE_TIME_ABS_VEC:
       AFSTime u_time_abs_vec<AFSVOL_TLV_TIME_MAX>;

    case AFSVOL_TLV_TYPE_TIME_REL_VEC:
       AFSRelTimestamp u_time_rel_vec<AFSVOL_TLV_UINT64_MAX>;

    case AFSVOL_TLV_TYPE_UUID:
       afsUUID u_uuid;
```

```
    case AFSVOL_TLV_TYPE_STRING:
       string u_string<AFSVOL_TLV_OPAQUE_MAX>;

    case AFSVOL_TLV_TYPE_VOL_DOW_USE:
       /* type defined later in this memo */
       AFSVol_stat_use_per_dow u_vol_dow_use;

    case AFSVOL_TLV_TYPE_OPAQUE:
       opaque u_opaque<AFSVOL_TLV_OPAQUE_MAX>;
   };

   /* registrar-controlled tag namespace */
   enum AFSVol_TLV_tag {
       AFSVOL_TLV_TAG_EOS = 0,
       AFSVOL_TLV_TAG_VOL_NAME = 1,
       AFSVOL_TLV_TAG_VOL_STATUS = 2,
       AFSVOL_TLV_TAG_VOL_IN_USE = 3,
       AFSVOL_TLV_TAG_VOL_ID = 4,
       AFSVOL_TLV_TAG_VOL_TYPE = 5,
       AFSVOL_TLV_TAG_VOL_CLONE_ID = 6,
       AFSVOL_TLV_TAG_VOL_BACKUP_ID = 7,
       AFSVOL_TLV_TAG_VOL_PARENT_ID = 8,
       AFSVOL_TLV_TAG_VOL_COPY_DATE = 9,
       AFSVOL_TLV_TAG_VOL_CREATE_DATE = 10,
       AFSVOL_TLV_TAG_VOL_ACCESS_DATE = 11,
       AFSVOL_TLV_TAG_VOL_UPDATE_DATE = 12,
       AFSVOL_TLV_TAG_VOL_BACKUP_DATE = 13,
       AFSVOL_TLV_TAG_VOL_SIZE = 14,
       AFSVOL_TLV_TAG_VOL_FILE_COUNT = 15,
       AFSVOL_TLV_TAG_VOL_QUOTA_BLOCKS = 16,
       AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY = 17,
       AFSVOL_TLV_TAG_VOL_STAT_USE_PER_DOW = 18,
       AFSVOL_TLV_TAG_VOL_STAT_READS = 19,
       AFSVOL_TLV_TAG_VOL_STAT_WRITES = 20,
       AFSVOL_TLV_TAG_VOL_STAT_FILE_SAME_AUTHOR = 21,
       AFSVOL_TLV_TAG_VOL_STAT_FILE_DIFFERENT_AUTHOR = 22,
       AFSVOL_TLV_TAG_VOL_STAT_DIR_SAME_AUTHOR = 23,
       AFSVOL_TLV_TAG_VOL_STAT_DIR_DIFFERENT_AUTHOR = 24,
       AFSVOL_TLV_TAG_VOL_TRANS_ID = 25,
       AFSVOL_TLV_TAG_VOL_TRANS_TIME = 26,
       AFSVOL_TLV_TAG_VOL_TRANS_CREATE_TIME = 27,
       AFSVOL_TLV_TAG_VOL_TRANS_RETURN_CODE = 28,
       AFSVOL_TLV_TAG_VOL_TRANS_ATTACH_MODE = 29,
       AFSVOL_TLV_TAG_VOL_TRANS_STATUS = 30,
       AFSVOL_TLV_TAG_VOL_TRANS_FLAGS = 31,
       AFSVOL_TLV_TAG_VOL_TRANS_LAST_PROC_NAME = 32,
       AFSVOL_TLV_TAG_VOL_TRANS_CALL_VALID = 33,
       AFSVOL_TLV_TAG_VOL_TRANS_READ_NEXT = 34,
```

```
        AFSVOL_TLV_TAG_VOL_TRANS_XMIT_NEXT = 35,
        AFSVOL_TLV_TAG_VOL_TRANS_LAST_RECV_TIME = 36,
        AFSVOL_TLV_TAG_VOL_TRANS_LAST_SEND_TIME = 37,
        AFSVOL_TLV_TAG_VOL_IN_SERVICE = 38,
        AFSVOL_TLV_TAG_VOL_BLESSED = 39,
        AFSVOL_TLV_TAG_VOL_RESTORED_FROM_ID = 40,
        AFSVOL_TLV_TAG_VOL_DESTROYED = 41,
        AFSVOL_TLV_TAG_VOL_NEEDS_SALVAGE = 42,
        AFSVOL_TLV_TAG_VOL_OFFLINE_MESSAGE = 43,
        AFSVOL_TLV_TAG_VOL_EXPIRATION_DATE = 44,
        AFSVOL_TLV_TAG_VOL_QUOTA_RESERVATION = 45,
        AFSVOL_TLV_TAG_VOL_STAT_USE_TODAY_DATE = 46,
        AFSVOL_TLV_TAG_VOL_STATE_ONLINE = 47,
        AFSVOL_TLV_TAG_VOL_STATE_AVAILABLE = 48,
        AFSVOL_TLV_TAG_VOL_STATE_EXPL = 49,
        AFSVOL_TLV_TAG_VOL_STATE_DAFS_RAW = 50,
        AFSVOL_TLV_TAG_VOL_STATE_OWNING_PROCESS = 51,
        AFSVOL_TLV_TAG_VOL_QUOTA_BLOCKS_STORED_LOCALLY = 52,
        AFSVOL_TLV_TAG_VOL_QUOTA_FILES = 53
    };


    struct AFSVol_TLV {
        afs_uint32 tlv_tag;
        afs_uint32 tlv_flags;
        AFSVol_TLV_value tlv_value;
    };

    struct AFSVol_TLV_query {
        AFSVol_TLV_tag tq_tag;
        AFSVol_TLV_value tq_qualifier;
    };

    struct AFSVol_TLV_store {
        AFSVol_TLV ts_tuple;
        AFSVol_TLV_value ts_qualifier;
    };

    typedef afs_uint64 AFSVol_TLV_TSV;
    typedef AFSVol_TLV_tag AFSVol_TLV_tag_vec<AFSVOL_TLV_TAG_MAX>;
    typedef AFSVol_TLV_query AFSVol_TLV_query_vec<AFSVOL_TLV_TAG_MAX>;
    typedef AFSVol_TLV AFSVol_TLV_vec<AFSVOL_TLV_TAG_MAX>;
    typedef afs_uint64 AFSVol_TLV_part_id_vec<AFSVOL_BULK_GETVOLUME_MAX>;
    typedef afs_uint64 AFSVol_TLV_vol_id_vec<AFSVOL_BULK_GETVOLUME_MAX>;
    typedef AFSVol_TLV_store AFSVol_TLV_store_vec<AFSVOL_TLV_TAG_MAX>;
    typedef afs_int32 AFSVol_TLV_result_vec<AFSVOL_TLV_TAG_MAX>;

    struct AFSVol_TLV_vol_list {
```

```
        afs_uint64 partId;
        AFSVol_TLV_Vol_id_vec * volIds;
    };

    typedef struct AFSVol_TLV_vol_list
    AFSVol_TLV_get_filter<AFSVOL_BULK_GETVOLUME_MAX>;


    proc GetVolumeTLVTags(
        IN AFSVol_TLV_tag offset,
        OUT AFSVol_TLV_tag_vec * tags,
        OUT AFSVol_TLV_TSV * tsv
    ) = XXX;

    proc GetOneVolumeTLV(
        IN afs_uint64 partId,
        IN afs_uint64 volId,
        IN AFSVol_TLV_query_vec * queries,
        OUT AFSVol_TLV_vec * tuples,
        OUT AFSVol_TLV_TSV * tsv
    ) = XXX;

    proc GetVolumesTLV(
        IN AFSVol_TLV_get_filter * filter,
        IN AFSVol_TLV_query_vec * queries,
        OUT AFSVol_TLV_TSV * tsv
    ) split = XXX;

    proc SetVolumeTLV(
        IN afs_int32 trans,
        IN AFSVol_TLV_TSV assert_tsv
        IN AFSVol_TLV_store_vec * tuples,
        OUT AFSVol_TLV_result_vec * results,
        OUT AFSVol_TLV_TSV * server_tsv
    ) = XXX;
```

Figure 7

Authors' Addresses

Thomas Keiser
Sine Nomine Associates
43596 Blacksmith Square
Ashburn, VA 20147
USA

Email: tkeiser@gmail.com

Steven Jenkins

Email: steven.jenkins@gmail.com

Andrew Deason (editor)
Sine Nomine Associates
43596 Blacksmith Square
Ashburn, Virginia 20147-4606
USA

Phone: +1 703 723 6673
Email: adeason@sinenomine.net