Network Working Group	T.E. Keiser
Internet-Draft	Sine Nomine
Intended status: Informational	June 02, 2011
Expires: December 04, 2011	

AFS-3 Rx RPC XDR Primitive Type Definitions
draft-keiser-afs3-xdr-primitive-types-00

Abstract

AFS-3 embeds a set of XDR primitive type definitions, which are referenced throughout the various AFS-3 protocol specifications. This memo defines the mapping between these AFS-3 primitive types, and the underlying XDR primitives.

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.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

AFS-3 [AFS1] [AFS2] is a distributed file system that has its origins in the VICE project [CMU-ITC-84-020] [VICE1] 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 [AFS3-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 In addition to using OpenAFS as a reference, IBM wrote and donated developer documentation that contains somewhat outdated specifications for the Rx protocol and all AFS-3 remote procedure calls, as well as a detailed description of the AFS-3 system architecture.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 Rx RPC protocol utilizes XDR [RFC4506] as its means of encoding RPC call and response payloads. While XDR provides type definitions for each popular bit-length integer, it does so using relatively ambiguous names (e.g., hyper). To improve readability, AFS-3 standards reference custom XDR types that embed the bit length within the type name. This memo standardizes those primitive types, as well as the encoding for the AFS-3 UUID.

1.1. Abbreviations

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

DCE - The Distributed Computing Environment

LSB - Least-Significant Bit

MSB - Most-Significant Bit

RPC - Remote Procedure Call

Rx - The Remote Procedure Call mechanism utilized by AFS-3

UUID - Universally Unique IDentifier

XDR - eXternal Data Representation

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. Primitive Integer Types

AFS-3 defines a number of special types which are direct mappings onto existing XDR types. Thus, these types are essentially XDR typedefs:

AFS-3 type name	->	XDR primitive type	[Reference]
char		int	RFC 4506 Section 4.1
unsigned char		unsigned int	RFC 4506 Section 4.2
afs_int8		int	RFC 4506 Section 4.1
afs_uint8		unsigned int	RFC 4506 Section 4.2
short		int	RFC 4506 Section 4.1
unsigned short		unsigned int	RFC 4506 Section 4.2
afs_int16		int	RFC 4506 Section 4.1
afs_uint16		unsigned int	RFC 4506 Section 4.2
afs_int32		int	RFC 4506 Section 4.1
afs_uint32		unsigned int	RFC 4506 Section 4.2
afs_int64		hyper	RFC 4506 Section 4.5
afs_uint64		unsigned hyper	RFC 4506 Section 4.5

AFS-3 common typedefs

3.1. char

This type is considered deprecated; future protocol specifications should reference the "afs_int8" type instead.

3.2. unsigned char

This type is considered deprecated; future protocol specifications should reference the "afs_uint8" type instead.

3.3. short

This type is considered deprecated; future protocol specifications should reference the "afs_int16" type instead.

3.4. unsigned short

This type is considered deprecated; future protocol specifications should reference the "afs_uint16" type instead.

3.5. 1- and 2-octet integer types

Please note that XDR uses a 4-octet alignment, and thus these 1- and 2-octet types will take 4 octets on the wire. Consequently, this is merely a way of defining data structures that have an optimized in-memory footprint, without imbuing any wire-encoding advantage.

4. afsUUID

AFS-3, being closely related to DCE, relies heavily upon a UUID type. The AFS-3 UUID type is identical to the DCE-variant, version 1 UUID type (see Appendix A of [dce_rpc]). The C data structure definition for such a UUID is as follows:

```
struct afsUUID {
    afs_uint32 time_low;
    afs_uint16 time_mid;
    afs_uint16 time_hi_and_version;
    char clock_seq_hi_and_reserved;
    char clock_seq_low;
    char node[6];
};
typedef struct afsUUID afsUUID;
```

An afsUUID type is XDR encoded on the wire as follows:

```
(MSB)                                                           (LSB)
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            time_low                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               0               |           time_mid            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               0               |      time_hi_and_version      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      {3}                      |      {1}      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      {3}                      |      {2}      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      {3}                      |    node[0]    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      {3}                      |    node[1]    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      {3}                      |    node[2]    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      {3}                      |    node[3]    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      {3}                      |    node[4]    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      {3}                      |    node[5]    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

    {1} clock_seq_hi_and_reserved
    {2} clock_seq_low
    {3} sign extended padding: 0, or 0xffffff depending on value
        of MSB in field to be sign-extended and padded
```

afsUUID XDR encoding

4.1. Encoding

XDR encoding an "afsUUID" type SHALL involve the following sequence of operations:

1. Encode "time_low" field as an XDR unsigned integer (afs_uint32)

2. Encode "time_mid" field as an XDR unsigned integer

3. Encode "time_hi_and_version" field as an XDR unsigned integer

4. Encode "clock_seq_hi_and_reserved" field as an XDR signed integer

5. Encode "clock_seq_low" field as an XDR signed integer

6. Encode "node[0]" field as an XDR signed integer

7. Encode "node[1]" field as an XDR signed integer

8. Encode "node[2]" field as an XDR signed integer

9. Encode "node[3]" field as an XDR signed integer

10. Encode "node[4]" field as an XDR signed integer

11. Encode "node[5]" field as an XDR signed integer

Many of the fields which are encoded in this data structure are smaller than four octets. In order to XDR encode these fields, they must first be resized. Since many of these fields are signed, this involves sign extension. This process depends upon the machine architecture. Virtually all machines in existence today utilize 2s-complement integer arithmetic, where this process merely involves padding with zeros if the MSB is zero or ones if the MSB is one.

4.2. Decoding

XDR decoding an "afsUUID" type SHALL involve the following sequence of operations:

1. Decode an XDR unsigned integer into the "time_low" field

2. Decode an XDR unsigned integer. If the integer is greater than 65535, the decoding SHALL fail. Copy the least-significant 16 bits into the "time_mid" field.

3. Decode an XDR unsigned integer. If the integer is greater than 65535, the decoding SHALL fail. Copy the least-significant 16 bits into the "time_hi_and_version" field.

4. Decode an XDR signed integer. If the integer is greater than 32767, or less than -32768, the decoding SHALL fail. Copy the least-significant 16 bits into the "clock_seq_hi_and_reserved" field.

5. Decode an XDR signed integer. If the integer is greater than 32767, or less than -32768, the decoding SHALL fail. Copy the least-significant 16 bits into the "clock_seq_low" field.

6. Decode an XDR signed integer. If the integer is greater than 127, or less than -128, the decoding SHALL fail. Copy the least-significant 8 bits into the "node[0]" field.

7. Decode an XDR signed integer. If the integer is greater than 127, or less than -128, the decoding SHALL fail. Copy the least-significant 8 bits into the "node[1]" field.

8. Decode an XDR signed integer. If the integer is greater than 127, or less than -128, the decoding SHALL fail. Copy the least-significant 8 bits into the "node[2]" field.

9. Decode an XDR signed integer. If the integer is greater than 127, or less than -128, the decoding SHALL fail. Copy the least-significant 8 bits into the "node[3]" field.

10. Decode an XDR signed integer. If the integer is greater than 127, or less than -128, the decoding SHALL fail. Copy the least-significant 8 bits into the "node[4]" field.

11. Decode an XDR signed integer. If the integer is greater than 127, or less than -128, the decoding SHALL fail. Copy the least-significant 8 bits into the "node[5]" field.

5. IANA Considerations

This memo includes no request to IANA.

6. AFS Assign Numbers Registrar Considerations

This memo includes no request to the AFS Assigned Numbers Registrar.

7. Security Considerations

This document merely describes various AFS-3 XDR types. Any protocol specification which uses these primitives types must ensure the security of the resulting XDR data streams, e.g., via prescription of a suitable Rx RPC security class.

8. References

8.1. Normative References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC4506]	Eisler, M., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.

8.2. Informative References

[CMU-ITC-84-020]	West, M.J., "VICE File System Services", CMU ITC Tech. Rep. CMU-ITC-84-020, August 1984.
[CMU-ITC-83-025]	Morris, J.H., Van Houweling, D. and K. Slack, "The Information Technology Center", CMU ITC Tech. Rep. CMU-ITC-83-025, 1983.
[AFS3-RX]	Zayas, E.R., "AFS-3 Programmer's Reference: Specification for the Rx Remote Procedure Call Facility", Transarc Corp. Tech. Rep. FS-00-D164, August 1991.
[VICE1]	Satyanarayanan, M., Howard, J.H., Nichols, D.A., Sidebotham, R.N., Spector, A.Z. and M.J. West, "The ITC Distributed File System: Principles and Design", Proc. 10th ACM Symp. Operating Sys. Princ. Vol. 19, No. 5, December 1985.
[AFS1]	Howard, J.H., "An Overview of the Andrew File System"", Proc. 1988 USENIX Winter Tech. Conf. pp. 23-26, February 1988.
[AFS2]	Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N. and M.J. West, "Scale and Performance in a Distributed File System", ACM Trans. Comp. Sys. Vol. 6, No. 1, pp. 51-81, February 1988.
[dce_rpc]	The Open Group, "CAE Specification, DCE 1.1: Remote Procedure Call", CAE C706, August 1997.

Appendix A. Base Type Definitions

```
typedef afs_int8  int;
typedef afs_uint8 unsigned int;
typedef afs_int16 int;
typedef afs_uint16 unsigned int;
typedef afs_int32 int;
typedef afs_uint32 unsigned int;
typedef afs_int64 hyper;
typedef afs_uint64 unsigned hyper;
```

Author's Address

Thomas Keiser Keiser Sine Nomine Associates 43596 Blacksmith Square Ashburn, VA 20147 USA Phone: +1 703 723 6673 EMail: tkeiser@sinenomine.net