N/A	A.P.D. Deason
Internet-Draft	Sine Nomine
Intended status: Informational	August 27, 2011
Expires: February 28, 2012	

Base Types for Time in AFS-3
draft-deason-afs3-type-time-03

Abstract

This document defines three types to be used in future AFS-3 Rx Remote Procedure Calls (RPCs) to represent time. Current AFS-3 RPCs represent time as 32-bit integers representing seconds. This is insufficient in both granularity and range, so new types to represent time are defined in this document to overcome these limitations.

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

All extant AFS-3 RPCs represent time as a 32-bit integer, as encoded by XDR in [RFC4506], which represents a number of seconds. For RPCs that specify an absolute time, this is the number of seconds that have passed since since midnight or 0 hour January 1, 1970 Coordinated Universal Time (UTC), not counting leap seconds. These time structures

will be unusable after January 2038, and are already insufficient to represent time with more granularity than one second.
This limited granularity creates inefficiencies in various parts of the AFS-3 protocol when it must be determined in what order two events have occurred (for example, whether or not a file was changed since the last time a volume has been backed up). When those two events have occurred during the same second, implementations must take a conservative assumption about which event occurred first, often resulting in unnecessary duplication or retransmission of data. In addition, metadata can be lost when files are copied to AFS from other filesystems that store file modification times with finer granularity than one second.
This document defines three new types to represent time to overcome these limitations: AFSAbsTime64, AFSRelTime64, and AFSAbsTime64Res. All of these support a much wider time range at a much higher granularity than the current time representations. AFSRelTime64 is to be used to represent times relative to some other event, and AFSAbsTime64 is to be used to represent absolute time stamps. AFSAbsTime64Res is to be used to represent a small range of absolute time, which is necessary for determining relative ordering of events, as described in Section 4.
All of these new types also have the additional benefit of providing standard type identifiers to be used when specifying absolute or relative time in AFS-3 RPC arguments and structures. Currently, all time values are just defined as "afs_int32" types in the XDR language definitions. This can make it confusing whether or not a field is an absolute time, relative time, or something else completely. Using the new standard time types will make this clear and unambiguous.

2. Conventions Used in this Document

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

3. Data Types

This document defines three new data types: AFSAbsTime64, AFSRelTime64, and AFSAbsTime64Res. All of these are encoded on the wire using the XDR standard described in [RFC4506], and are described using the XDR language specification therein.

3.1. AFSAbsTime64

The new AFSAbsTime64 type is represented as an XDR-encoded 64-bit signed integer representing the timestamp. It is defined as thus in XDR:

```
typedef hyper AFSAbsTime64;
```

The AFSAbsTime64 type represents time relative to midnight or 0 hour January 1, 1970 Coordinated Universal Time (UTC), represented in increments of 100 nanoseconds (ns). If the value is greater than zero, the value represents the amount of time that has passed since midnight January 1, 1970 UTC, excluding any leap seconds. If the value is less than zero, the value represents the amount of time before midnight January 1, 1970 UTC.
For example, to represent the time 60 seconds after midnight on January 1, 1970 UTC, the value of the field would be 600000000 (600 million). To represent the time 60 seconds before midnight January, 1970 UTC, the value of the field would be -600000000 (negative 600 million).
This type can represent any time in the year 27258 BCE through any time in the year 31196 CE with 100-ns granularity. For timestamps before 1972, see the notes in Section 6.

3.2. AFSRelTime64

The new AFSRelTime64 type has same representation on the wire as AFSAbsTime64 in Section 3.1:

```
typedef hyper AFSRelTime64;
```

The AFSRelTime64 type represents an amount of time that has passed since some other event, represented in increments of 100 nanoseconds (ns). The event to which this time is relative is unspecified, and can be anything; it must be specified by the RPC or structure that defines a field of the AFSRelTime type.
Values greater than 0 represent dates that occur after the relative event, and values less than 0 represent dates that occur before the relative event.
For example, to represent the time 5 seconds before some other event, the value of the timestamp field would be -50000000 (negative 50 million).

3.3. AFSAbsTime64Res

The new AFSAbsTime64Res type is represented as an XDR-encoded structure on the wire, containing an AFSAbsTime64 and a 32-bit unsigned integer representing the resolution. It is defined as thus in XDR:

```
struct AFSAbsTime64Res {
    AFSAbsTime64 timestamp;
    unsigned int resolution;
};
```

The AFSAbsTime64Res structure represents the amount of time that has passed since midnight or 0 hour January 1, 1970 Coordinated Universal Time (UTC), excluding any leap seconds. The value of the timestamp field is this amount of time represented as described in Section 3.1.

The resolution field represents the resolution of the time source from which the timestamp was obtained. The value of the resolution field is the difference between the represented time, and another time after the represented time that is guaranteed to be after the actual time at which the event in question occurred.
In other words, let X be the time that some event occurred, Y be the value of the timestamp field in an AFSAbsTime64Res structure, and Z be the value of the resolution field. To construct an AFSAbsTime64Res structure that represents X, the values of the timestamp and resolution field MUST be specified such that Y <= X < Y + Z. Typically the value of the resolution field will just be the resolution of the time source from which the timestamp was obtained, if the resolution of the time source is constant.
For example, to represent the time 60 seconds after midnight on January 1, 1970, the value of the timestamp field would be 600000000 (600 million) as described in Section 3.1. If this time stamp was obtained from a source that only represents time in seconds, the next representable time is 61 seconds after midnight on January 1, 1970, so it is guaranteed that the event in question occurred before that time (otherwise the time source would give us a time of 61 seconds). So the value of the resolution field should be 10000000 (10 million). In effect, this AFSAbsTime64Res structure represents an event that occurred at or after 60 seconds after January 1, 1970, but occurred before 61 seconds after January 1, 1970.
For more details about the resolution field (including the motivation for its existence), see Section 4.

3.3.1. Resolution Assumptions

If the resolution field has the value of 0, the resolution of the specified timestamp is unknown. If an implementation has absolutely no mechanism to determine the resolution of a time source when creating a time stamp, it MUST specify a resolution of 0. If an implementation needs to use an AFSAbsTime64Res value for calculating event ordering, and the resolution is 0, it SHOULD assume a resolution of 1 second, and round the timestamp down to the nearest second.
An AFS-3 implementation MUST NOT ever specify a resolution greater than 1 second (10000000 100-ns increments). Implementations of the AFS-3 protocol that exist prior to the introduction of the new time types in this document assume that the time resolution is 1 second, and may not behave correctly with time sources that are less granular than 1 second. No systems nor file formats that are related to any AFS-3 implementation are known that do not have at least 1-second granularity, so adhering to this should not be a problem.
If an implementation receives an AFSAbsTime64Res structure with a granularity coarser than 1 second, it MUST treat it as an invalid time representation. What that entails depends on the context, but the AFSAbsTime64Res value MUST NOT be used for any calculations and SHOULD be immediately discarded. Typically an RPC will raise some kind of

error in this condition, but the exact behavior is up to the relevant RPC or other operation.

4. Time Resolution

The new type AFSAbsTime64Res includes information about the resolution or granularity of the time it represents. The reason for including this information may not be immediately clear, so this section provides some information on why this information is beneficial.

4.1. Sources of Differing Time Resolutions

All current AFS-3 implementations represent time as a 32-bit integer on the wire, and so it is common for implementations to internally represent time as 32-bit integers, with 1-second granularity. As such, when implementations support the time types defined in this document, it is likely that there will be a period of time where implementations cannot store or represent time with greater granularity than 1-second, even though the protocol allows them to do so.
In addition, due to technical restrictions of various platforms, or other sources of time (such as file formats), implementations may be only able to transmit time information in certain granularities. For example, the operating system that an AFS-3 Volume Server implementation runs on may only be able to retrieve the current time in increments of 1 millisecond. Or, an AFS-3 Volume Server implementation may be reading the time information from a file, and the file only represents time in increments of 1 second.
From this, it is clear that implementations will send time of various granularity when communicating with other services and clients in AFS-3, and the granularity of time may vary even within the same implementation process (depending on from where it is obtaining the time).

4.2. Relevance to AFS-3

Timestamps are sometimes used in AFS-3 to establish a relative non-strict total ordering of events. That is, given the events X and Y, we must determine whether X or Y occurred first, or if they occurred at approximately the same time. This primarily occurs in volume operations when incremental data is sent, and exactly what incremental data is sent is determined by the timestamps of other volumes. If we get the ordering wrong, problems with data inconsistency can occur. If we conservatively determine that two events occurred at the same time when we could have correctly made the determination that one occurred before the other, inefficiencies arise.
In order to be able to order such events, then, we must know the resolution of the time value that is stored. This is so we know the earliest possible time that the event occurred, and the latest possible time that the event occurred.

4.3. When to Include Resolution Information

There are two time types defined in this document to represent an absolute timestamp: AFSAbsTime64 and AFSAbsTime64Res. The only difference between these two types is that AFSAbsTime64Res includes resolution information, and AFSAbsTime64 does not. This section serves to guide designers of future AFS-3 RPCs in what circumstances each type should be used.

When deciding whether to use AFSAbsTime64 or AFSAbsTime64Res in a structure field or RPC argument, the first determination that must be made is whether the timestamp value will or could ever be used to make ordering decisions by an AFS-3 service. If it will or could, then the argument or field should be of the AFSAbsTime64Res type.

Otherwise, the determination should be made whether or not any service (possibly unrelated to AFS-3) may realistically make ordering decisions based on the field or argument. If it may, then the AFSAbsTime64Res type should be considered; otherwise, the AFSAbsTime64 type should be used.

For example, as mentioned in Section 4.2, the AFS-3 Volume Service and clients make ordering decisions based on timestamps related to when volume operations have occurred. So, timestamp fields related to volume operations should probably use the AFSAbsTime64Res type.

However, when a timestamp is used to represent the modification time of a file in AFS, that timestamp is not used by AFS-3 services to make any ordering decisions. While it is possible that some software unrelated to AFS-3 may try to make ordering decisions based off of that timestamp, it is unlikely to be able to do so reliably. This is because file modification timestamps can usually be set to any time by humans, and any time resolution information stored is usually not available to programs that are not AFS-3-aware. And in general, AFS-3 file metadata is not intended to be a general-purpose distributed synchronization mechanism. So, file modification timestamps should probably use the AFSAbsTime64 type.

5. Resolution Limitations

The types specified in this document are limited to 100-nanosecond resolution or coarser. There are other systems which may interact with AFS-3 that have finer resolution; for example, the NFSv4 nfstime4 structure in Section 2.2 of [RFC3530] and the timespec structure in [POSIX] both allow for timestamps with a resolution of 1-ns. Because of this, there are inherent problems with interacting with such systems.

It is believed that for a large majority of use cases, timestamps with resolution finer than 100-nanoseconds are not necessary, and so the types defined in this document should be sufficient. However, there may be use cases in which resolution finer than 100-ns is required. In addition, there are also several use cases where the legacy 32-bit timestamps are adequate, and the additional space overhead of the types defined by this document may be considered unnecessary overhead.

This document does not accommodate for those cases, but this document does not restrict AFS-3 to only use the time types defined in this document. It is recommended that future AFS-3 RPCs are designed such that they make take advantage of several different time types of varying resolutions, so that such use cases can be accommodated while not sacrificing the space efficiency for the common case addressed by the time types defined in this document. Any such method of accommodating for different time types are left up to the individual RPCs or wire structures, and are not discussed here.

6. Times Before UTC

The absolute time types defined in this document are specified as relative to midnight January 1, 1970 UTC, excluding leap seconds, and times far earlier than that are also representable. It is worthy to note that UTC did not exist until January 1, 1972, and so times before 1972 specified as UTC are technically meaningless. However, it is convenient to assume that UTC has existed for all eternity. For all times before 1972, we represent time as if UTC has always existed, using the obvious backwards projection of the current UTC time zone and Gregorian calendar rules.

7. Converting Time Types

In general, when converting an AFSAbsTime64 or AFSAbsTime64Res value to some other type that has granularity coarser than 100 ns granularity, the resulting value MUST always be rounded down to the nearest lower increment of the resultant type. When converting to the POSIX time_t type, for example, the AFSAbsTime or AFSAbsTime64Res value MUST be rounded down to the nearest lower second. When converting to the POSIX struct timeval type, the value MUST be rounded down to the nearest lower microsecond.
When converting to or from the Microsoft FILETIME format, a constant value must be added or subtracted, since FILETIME specifies time relative to midnight 1 January 1601 UTC, but AFSAbsTime64 and AFSAbsTime64Res specify time relative to midnight 1 January 1970 UTC. When converting from an AFSAbsTime64 to a Microsoft FILETIME, the value 116444736000000000 must be subtracted from the AFSAbsTime64 value. When converting to an AFSAbsTime64 from a Microsoft FILETIME, the value 116444736000000000 must be added to the FILETIME value.
It is also important to keep in mind that when converting from AFSAbsTime64Res to time_t or FILETIME types, strictly speaking the AFSAbsTime64Res structure represents two times: the beginning and end time. So it is impossible to accurately convert an AFSAbsTime64Res structure to a single time value. This is often unavoidable when converting to legacy AFS-3 interfaces, or interfaces unrelated to AFS-3, however, and when only one time value is given to convert to, implementations MUST specify the beginning time (that is, the time represented by the timestamp field).

7.1. Special Cases

Extant AFS-3 RPCs often use a timestamp of 0 to represent a special meaning. That is, a timestamp of 0 often does not indicate midnight 1 January 1970 UTC, but may represent a logical value of "negative infinity", or indicates some special meaning that is specific to that RPC. The value of 0 was often just used because it is the earliest representable time for a 32-bit unsigned integer.

Any such special meanings must be specified by the RPC in question, and this document assigns no special meaning to the value of 0 for any of the types defined in this document. However, this document recommends that any future RPCs keep the special meaning of the 0 timestamp, if the RPC is replacing an RPC that previously had a special meaning for timestamp 0, even though that is no longer the earliest representable time. If the new RPC uses an AFSAbsTime64Res argument, the resolution field should be 0, as well.

For example, say there is an AFS-3 RPC called AFSFoo that accepts an afs_uint32 absolute timestamp argument, and it specifies that a timestamp of 0 represents some special case. Let another RPC, AFSFoo64, define an RPC that is identical to AFSFoo except that it accepts an AFSAbsTime64Res parameter. AFSFoo64 should specify that a caller should specify an AFSAbsTime64Res with timestamp 0, resolution 0, to indicate the special case previously indicated by giving a 32-bit timestamp of 0.

7.2. Sample Code

Sample C code is provided here to convert AFSAbsTime64, AFSRelTime64, and AFSAbsTime64Res values to and from FILETIME and POSIX time_t values where appropriate. Also provided is a function to compare two AFSAbsTime64Res values, and functions to add an AFSRelTime64 to an AFSAbsTime64Res and AFSAbsTime64.

The conversion functions follow the recommendations in Section 7.1. So, a time_t with the value of 0 will be converted to an AFSAbsTime with the value of 0, or an AFSAbsTime64Res value of timestamp 0, resolution 0. AFSAbsTime64Res structures with a resolution of 0 (and a non-zero timestamp) are treated as having an effective resolution of 1 second, as suggested in Section 3.3.1.

These functions do not handle overflow, underflow, or other errors, and are just guidelines for the general conversion algorithms.

```
#define AFSTIME64_WINNT_SHIFT 116444736000000000ULL
#define AFSTIME64_POSIX_SCALE 10000000U
#define AFSTIME64_DEFAULT_RES 10000000U

void
timet_to_AFSAbsTime64(time_t t, AFSAbsTime64 *atsp)
{
    *atsp = (t * AFSTIME64_POSIX_SCALE);
}
void
AFSAbsTime64_to_timet(AFSAbsTime64 ats, time_t *tp)
{
    *tp = (ats / AFSTIME64_POSIX_SCALE);
}
void
timet_to_AFSRelTime64(time_t t, AFSRelTime64 *artsp)
{
    *artsp = (t * AFSTIME64_POSIX_SCALE);
}
void
AFSRelTime64_to_timet(AFSRelTime64 arts, time_t *tp)
{
    *tp = (arts / AFSTIME64_POSIX_SCALE);
}
void
timet_to_AFSAbsTime64Res(time_t t, AFSAbsTime64Res *atp)
{
    timet_to_AFSAbsTime64(t, &atp->timestamp);
    if (t == 0) {
        atp->resolution = 0;
    } else {
        atp->resolution = AFSTIME64_POSIX_SCALE;
    }
}
void
AFSAbsTime64Res_to_timet(AFSAbsTime64Res *atp, time_t *t1, time_t *t2)
{
    unsigned int res = atp->resolution;
    if (res == 0) {
        res = AFSTIME64_DEFAULT_RES;
    }
    AFSAbsTime64_to_timet(atp->timestamp, t1);
    AFSAbsTime64_to_timet(atp->timestamp + res, t2);
}
void
FILETIME_to_AFSAbsTime64(FILETIME *ftp, AFSAbsTime64 *atsp)
{
    ULARGE_INTEGER uli;
    uli.LowPart = ftp->dwLowDateTime;
```

```
    uli.HighPart = ftp->dwHighDateTime;
    *atsp = uli.QuadPart - AFSTIME64_WINNT_SHIFT;
}
void
AFSAbsTime64_to_FILETIME(AFSAbsTime64 ats, FILETIME *ftp)
{
    ULARGE_INTEGER uli;
    uli.QuadPart = ats + AFSTIM64_WINNT_SHIFT;
    ftp->dwLowDateTime = uli.LowPart;
    ftp->dwHighDateTime = uli.HighPart;
}
void
AFSAbsTime64Res_to_FILETIME(AFSAbsTime64Res *atp, FILETIME *ft1,
                            FILETIME *ft2)
{
    unsigned int res = atp->resolution;
    if (res == 0) {
        res = AFSTIME64_DEFAULT_RES;
    }
    AFSAbsTime64_to_FILEMTIME(atp->timestamp, ft1);
    AFSAbsTime64_to_FILEMTIME(atp->timestamp + res, ft2);
}
void
AFSAbsTime64Res_add_AFSRelTime64(AFSAbsTime64Res *atp,
                                 AFSRelTime64 arts)
{
    atp->timestamp += arts;
}
void
AFSAbsTime64_add_AFSRelTime64(AFSAbsTime *atsp,
                              AFSRelTime arts)
{
    *atsp += arts;
}
int
cmp_AFSAbsTime64Res(AFSAbsTime64Res *atp1, AFSAbsTime64Res *atp2)
{
    unsigned int res1 = atp1->resolution;
    unsigned int res2 = atp2->resolution;
    if (res1 == 0) {
        res1 = AFSTIME64_DEFAULT_RES;
    }
    if (res2 == 0) {
        res2 = AFSTIME64_DEFAULT_RES;
    }
    if (atp1->timestamp + res1 <= atp2->timestamp) {
        return -1;
    }
    if (atp2->timestamp + res2 <= atp1->timestamp) {
```

```
        return 1;
    }
    return 0;
}
```

8. Security Considerations

This memo raises no security issues.

9. IANA Considerations

This document makes no request of the IANA.

10. Acknowledgements

The author thanks Simon Wilkinson and Jeffrey Altman for some background text, Jeffrey Altman, David Boyes, Tom Keiser, and Simon Wilkinson for discussion on the problem of time resolution, Steven Jenkins for discussion on interoperability concerns, Russ Allbery for input on UTC, leap seconds, and dates before 1970, and the general membership of the afs3-standardization@openafs.org mailing list for general discussion on the balance between time granularity and field width overhead.

11. References

11.1. Normative References

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

11.2. Informative References

[RFC3530]	Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M. and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003.
[RFC4506]	Eisler, M., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.
[POSIX]	Institute of Electrical and Electronics Engineers , "IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) Base Specifications, Issue 7 ", IEEE Std 1003.1-2008, 2008.

Author's Address

Andrew Deason Deason Sine Nomine Associates 43596 Blacksmith Square Ashburn, Virginia 20147-4606 USA Phone: +1 703 723 6673 EMail: adeason@sinenomine.net