The ATM Forum
Technical Committee

ATM Name System Specification Version 1.0

af-saa-0069.000

November, 1996

ATM Name System Specification Version 1.0
af-saa-0069.000

The information in this publication is believed to be accurate as of its publication date. Such information is subject to change without notice and the ATM Forum is not responsible for any errors. The ATM Forum does not assume any responsibility to update or correct any information in this publication. Notwithstanding anything to the contrary, neither The ATM Forum nor the publisher make any representation or warranty, expressed or implied, concerning the completeness, accuracy, or applicability of any information contained in this publication. No liability of any kind shall be assumed by The ATM Forum or the publisher as a result of reliance upon any information contained in this publication.

The receipt or any use of this document or its contents does not in any way create by implication otherwise:

- Any express or implied license or right to or under any ATM Forum member company's patent, copyright, trademark or trade secret rights which are or may be associated with the ideas, techniques, concepts or expressions contained herein; nor

- Any warranty or representation that any ATM Forum member companies will announce any product(s) and/or service(s) related thereto, or if such announcements are made, that such announced product(s) and/or service(s) embody any or all of the ideas, technologies, or concepts contained herein; nor

- Any form of relationship between any ATM Forum member companies and the recipient or user of this document.

Implementation or use of specific ATM standards or recommendations and ATM Forum specifications will be voluntary, and no company shall agree or be obliged to implement them by virtue of participation in The ATM Forum.

The ATM Forum is a non-profit international organization accelerating industry cooperation on ATM technology. The ATM Forum does not, expressly or otherwise, endorse or promote any specific products or services.

NOTE: The user's attention is called to the possibility that implementation of the ATM interoperability specification contained herein may require use of an invention covered by patent rights held by ATM Forum Member companies or others. By publication of this ATM interoperability specification, no position is taken by The ATM Forum with respect to validity of any patent claims or of any patent rights related thereto or the ability to obtain the license to use such rights. ATM Forum Member companies agree to grant licenses under the relevant patents they own on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license. For additional information contact:

The ATM Forum
Worldwide Headquarters
2570 West El Camino Real
Suite 304
Mountain View, CA 94040
Tel: +1-415-949-6700
Fax: +1-415-949-6705

Contents

1. Introduction

ATM applications require various types of directory services. Directory services can be universal, that is, used by more than one application, or they may be application specific. An example of a universal directory service is finding an ATM address corresponding to the name of an ATM end system. An example of an application specific service is finding the providers of specific types of services (for example, Video on Demand servers).

ATM Name System (ANS) is an extension to the Internet Engineering Task Force's (IETF) Domain Name System (DNS). However, rather than running on top of TCP/IP or UDP/IP like most DNS implementations, ANS is a native ATM application. As such, ANS uses an API whose semantics are specified in [ATMNAS] to gain access to the ATM network via ATM Forum UNI signaling. The ANS record formats and DNS functions are intended to be independent of the version of the underlying ATM signaling. See [ATMNAS] for information on the versions of the UNI signaling that are applicable to ANS.

ANS clients use ATM SVCs to communicate to their ANS servers. ANS version 1.0 servers use ATM SVCs or UDP to communicate ANS queries and replies to other ANS servers. ANS version 1.0 servers use TCP to communicate reliable zone transfer messages.

Basic name services are modeled on the Domain Name System [RFC1034, RFC1035]. Some extensions that may be considered for version 2.0 are security extensions, dynamic update capability, mechanisms for prompt notification of zone changes, and incremental zone transfer capability.

The basic directory services are domain name to ATM address translation and ATM address to domain name translation. The domain name to ATM address translation is performed under existing top level domain names. The ATM address to domain name translation is done using a new domain name of ATMA.INT. However, information other than ATM names and addresses may be added to the ANS data base. The ATM Forum (ATMF) supports rapid deployment of the essential name services and provides infrastructure for higher level directory services.

This specification is version 1.0 of the ATM Name System (ANS) and includes the basic functionality as defined in [RFC1034] and [RFC1035]. Since ANS is based on DNS, the SAA/DS group decided to refer to [RFC1034] and [RFC1035] rather than repeat information contained in these RFCs in this specification. Implementers will need to refer to these documents.

2. Terms and Definitions

See [RFC1034] and [RFC1035] for DNS terms and definitions.

3. Overview of ANS

See [RFC1034] and [RFC1035] for an overview of DNS.

4. Logical models

See section 2.2 of [RFC1035] for common DNS configurations.

5. Database elements

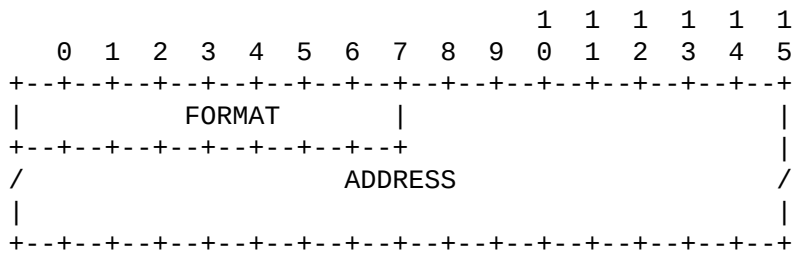
5.1 Resource Record Definitions

ATM specific resource records (RR) conform to the top level RR format and semantics as defined in Section 3.2.1 of [RFC1035] and belong to the RR CLASS Internet (IN).

5.2 ATM Specific Resource Records

This specification defines one ATM specific resource record, ATMA RR, that is used to map domain names to ATM addresses. The ATMA RR is defined with the mnemonic “ATMA” and TYPE code 34 (decimal). ATM address lookup is analogous to IP address lookup. A query is generated by the resolver requesting ATMA RRs for the provided domain name.

ATMA RR's have the following RDATA format:



The fields have the following meaning:

- FORMAT: One octet that indicates the format of ADDRESS. The two possible values for FORMAT are value 0 indicating ATM End System Address (AESA) format and value 1 indicating E.164 format.

- ADDRESS: Variable length string of octets containing the ATM address of the node to which this RR pertains.

  When the format value is 0, indicating that the address is in AESA format, the address is coded as described in ISO 8348/AD 2 using the preferred binary encoding of the ISO NSAP format. When the format value is 1, indicating that the address is in E.164 format, the Address/Number Digits appear in the order in which they would be entered on a numeric keypad. Digits are coded in IA5 characters with the leftmost bit of each digit set to 0. This ATM address appears in ATM End System Address Octets field (AESA format) or the Address/Number Digits field (E.164 format) of the Called party number information element [ATMUNI3.1]. Subaddress information is intentionally not included because E.164 subaddress information is used for routing.

ATMA RRs cause no additional section processing.

Other ATM specific resource records can be added in future releases of this specification.

5.3 ATM Address to Domain Name Mapping

The PTR RR is defined in [RFC1035]. This RR is typically used under the "IN-ADDR.ARPA" domain to map from IPv4 addresses to domain names.

Similarly, the PTR RR is used to map from ATM addresses to domain names under the "ATMA.INT" domain. A domain name is generated from the ATM address according to the rules described below. A query is sent by the resolver requesting a PTR RR for the provided domain name.

A domain name is generated from an AESA formatted ATM address by concatenating from left to right the following substrings and separating them by a "." character from each other:

- the SEL field
- the ESI field
- the digits of the HO-DSP field in reverse order and separated by a "." character from each other
- the DCC, ICD, or E.164 field
- the AFI field
- the top level subdomain "AESA.ATMA.INT."

For example, the domain name used in the reverse lookup for the ATM address

```
39246f000e7c9c0312000100010000 1234567800
```

would appear as

```
00.000012345678.1.0.0.0.1.0.0.0.2.1.3.0.c.9.c.7.e.0.0.0.246f.39.AE
SA.ATMA.INT.
```

A domain name is generated from an E.164 formatted ATM address by concatenating from left to right the following substrings and by separating them with a "." character.

- the digits of the country code national significant numbers in reverse order and separated by a "." character from each other
- the top level subdomain "E164.ATMA.INT."

For example, the domain name used in the reverse lookup for the E.164 number

```
3584001234567
```

would appear as

```
7.6.5.4.3.2.1.0.0.4.358.E164.ATMA.INT.
```

Implementation note: For sanity's sake user interfaces should be designed to allow users to enter ATM addresses using their natural order, that is, as they are typically written on paper. Also, arbitrary “.”s should be allowed (and ignored) on input.

5.4 Example Master File Format

This section describes an example master file format. Note that other formats are possible.

The format of AESA RRs in Master Files conforms to Section 5, "Master Files," of [RFC1035].

The RDATA section of an A line in a master file is expressed as follows:

AESA formatted ATM address: a string of hexadecimal digits. A “.” character can be used to separate any two digits for readability. The string is case insensitive. For example:

```
39.246f.00.0e7c9c.0312.0001.0001.000012345678.00
```

E.164 formatted ATM address: a “+” character followed by a string of decimal digits that form an international E.164 number. A “.” character can be used to separate any two digits for readability. For example:

```
+358.400.1234567
```

Below are examples of the use of ATMA and PTR RRs in Master Files to support name to ATM address and ATM address to name mapping.

```
;;;;;;;
;;;;;;; Master File for domain data.example.com.
;;;;;;;

@       IN    SOA    name1.data.example.com.  name4.data.example.com. (
                                   1994041800   ; Serial  - date
                                   1800         ; Refresh - 30 minutes
                                   300          ; Retry   - 5 minutes
                                   604800       ; Expire  - 7 days
                                   3600 )       ; Minimum - 1 hour
        IN    NS     name1.data.example.com.
        IN    NS     ns.example.com.
;
$ORIGIN data.example.com.
salmon IN     ATMA   39.246f.000e7c9c031200010001.000012345678.00
char   IN     ATMA   39.246f.000e7c9c031200010001.000023456789.00

;;;;;;;
;;;;;;; Master File for reverse mapping of ATM addresses under the
;;;;;;; prefix 39.246f.000e7c9c031200010001
;;;;;;;

@       IN    SOA    name1.data.example.com.  name4.data.example.com. (
                                   1994041800   ; Serial  - date
                                   1800         ; Refresh - 30 minutes
                                   300          ; Retry   - 5 minutes
                                   604800       ; Expire  - 7 days
                                   3600 )       ; Minimum - 1 hour
        IN    NS     name1.data.example.com.
        IN    NS     ns.example.com.
;
$ORIGIN 1.0.0.0.1.0.0.0.2.1.3.0.c.9.c.7.e.0.0.0.246f.39.AESA.atm.int.
00.000012345678       IN      PTR     salmon.data.example.com.
00.000012345679       IN      PTR     char.data.example.com.
```

6. DNS Protocol in ATM Environment

The DNS assumes that protocol messages will be transmitted as datagrams or in a byte stream carried by a virtual circuit. Datagrams are preferred for queries and responses due to their lower overhead. Zone refresh activities (query type AXFR) must use reliable virtual connections.

Since ATM networks do not support datagram services, ANS queries and responses must be carried over ATM virtual connections established between the clients and the servers. In version 1.0, the servers use TCP to reliably communicate among themselves.

6.1 Recursive Processing

By default, ANS clients, when they make queries, request recursive processing of the query (by setting the RD bit to 1 in the header of the query). ANS name servers must support recursion. If the server gives the client a referral, then it can happen that the client does not have ATM connectivity to the next server.

6.2 Client Initialization

When a client becomes operational, initially it needs to find the ATM addresses of one or more ANS servers. ANS clients use one of two mechanisms to locate the ATM address of ANS servers:

- Get the ANS server addresses via ILMI.

- Use a well-known ANS Address.

Getting ANS server addresses via ILMI is accomplished by defining a new type value for the atmfSrvcRegTypes ILMI object that denotes ANS server:

```
-- ANS Server
atmfSrvcRegAnsServer    OBJECT IDENTIFIER ::= { atmfSrvcRegTypes 2 }
```

If an ANS server address cannot be obtained from ILMI or if the client is unable to establish a VC to any ILMI supplied server address, then a well-known address may be used. The well known address is 02C5007900000000000000000000.00A03E000002.00.

6.3 Connection Usage

After learning the ATM address of one or more ANS servers, an ANS client may setup a connection to any one of the servers for sending ANS requests and receiving ANS responses. The client sets up and releases the connection dynamically using the UNI signaling protocol accessed through an API whose semantics are specified by [ATMNAS]. The client should time-out after some period of inactivity and release this connection. The server may release an inactive connection so that the server can provide services to other clients. Neither the server nor the client shall attribute any special significance to a disconnection indication.

7. Native ATM Services Interface

The ANS client and ANS server access the ATM network via an API whose semantics are specified by [ATMNAS]. The Native ATM Services connection

attributes used for communications between an ANS client and an ANS server are shown in the table below:

Attribute Name	Attribute Value
AAL_TYPE	AAL type 5
AAL5_FWD_MAX_SDU	512 bytes
AAL5_BAK_MAX_SDU	512 bytes
AAL5_SSCS_TYPE	Null
FWD_PCR_CLP1	line rate in cells per second
BAK_PCR_CLP1	line rate in cells per second
BEST_EFFORT	best effort requested
BEARER_CLASS	class X (BCOB-X)
TRAFFIC_TYPE	no indication
TIME_REQ	no indication
CLIPPING_IND	not susceptable to clipping
CONNECT_CONFIG	point-to-point connection
APPL_ID_TYPE	vendor-specific application ID
APPL_ID	The OUI (BHLI octets 6-8) is 0x00A03E. The application ID (BHLI octets 9-12) is 0x00000001.
CALLED_ADDR_FORMAT	either private or public address format
CALLED_ADDR	ATM address of DNS server
CALLED_SUBADDR_TYPE	private subaddress format, if required
CALLED_SUBADDR	ATM subaddress of DNS server, if required
FWD_QOS_CLASS	QoS class 0
BAK_QOS_CLASS	QoS class 0
SSCS	Null

The following elements form a SAP address for the DNS server:

- ATM address, including the selector byte
- DNS application identification (carried in the BHLI information element)

Note that both layer-2 protocol identification and layer-3 protocol identification are ABSENT.

8. ATM Directory Service Interface

The basic API functions for access to ATM directory services are expected to be made through calls to an API that includes a service to obtain a ATM address given a domain name and a service to obtain a domain name given an ATM address. An example of an API that could be used as a starting point is the one provided by the UNIX operating system [BSD4.3] and described in informative Appendix B. The example API provides `gethostbyname_atmnsap` and `gethostbyname_E164` services that accept host names as input and returns ANS database information about those host names. The returned information includes the host address type, host address length and host address, and may include other information such as a list of alias names for that host. The example API also provides a `gethostbyaddr` service that expects a host address type, host address length and host address as input. This service returns at least the host address name and optionally may return other ANS information as well.

9. References

[ATMNAS]
ATM Forum Technical Committee, “Native ATM Services: Semantic Description, Version 1.0”, af-saa-0048-000, ATM Forum, January, 1996

[ATMUNI3.1]
ATM Forum Technical Committee, “ATM User-Network Interface (UNI) Specification, Version 3.1”, ISBN 0-13-393828-X, ATM Forum, 1994

[BSD4.3] S. J. Leffler, M. K. McKusick, M. Karels,
The Design and Implementation of the 4.3 BSD Unix Operating System, 1989

[RFC1034]
P. Mockapetris, "Domain Names - Concepts and Facilities", RFC USC/Information Sciences Institute, November 1987.

[RFC1035]
P. Mockapetris, "Domain Names - Implementation and Specification", RFC 1035, USC/Information Sciences Institute, November 1987.

[RFC1700]
J. Reynolds and J. Postel, "Assigned Numbers", RFC 1700, ISI, October 1994.

[RFC1706]
B. Manning and R. Colella, "DNS NSAP Resource Records", RFC 1706, ISI/NIST, October 1994.

Appendix A: Requirements

This informative appendix captures the SAA/DS working group's requirements.

A.1 Basing of ANS on DNS

The ANS is based on the Domain Name Service (DNS) [RFC1034], [RFC1035]. Subsequent versions may include IETF enhancements for security, dynamic updates, prompt notification of zone changes and incremental zone transfer capability extensions. There are many reasons to use DNS as the basis for ANS.

A.1.1. DNS is widely implemented in end systems.
A.1.2. DNS domain registration procedure has been implemented in most countries.
A.1.3. DNS domain names are well known by the user community.
A.1.4. DNS can scale well if the name space is properly assigned. This satisfies the scalability requirement given in A.2.
A.1.5. DNS implementations are efficient in terms of CPU and memory usage. This satisfies the efficiency requirement given in A.2.
A.1.6. DNS is resilient via the secondary server mechanism. This satisfies the no single point of failure requirement given in A.2.
A.1.7. DNS has low latency of queries. This satisfies the requirement given in 2.1.4.
A.1.8. DNS extensions for dynamic updates and security are being worked on.
A.1.9. DNS can be easily augmented with new database objects for ATM.
A.1.10. DNS allows dual hosts to choose between IP and native ATM.

A.2 ANS requirements

Several requirements for the directory service function that apply to ANS are:

- No single point of failure (resilience)
- efficient implementation including storage efficiency
- Hierarchical naming structure (scalability)
- low latency of queries
- Toleration of temporary inconsistency, that is, that such inconsistency does not damage the information in ANS.
- The service may sometimes be transiently unavailability, for example, "a busy signal"

Other ANS requirements include:

- Independence between non-ATM network and transport layer protocols

ANS is to be used by ATM applications. These applications may not necessarily implement non-ATM network or transport protocols. Most current DNS implementations carry DNS protocol messages on top of UDP/IP and TCP/IP. This document specifies the operation of DNS protocols using native ATM services instead of TCP/IP.

- Use of an existing naming scheme

  ANS should exploit an already existing naming scheme in the same way as ATM addressing utilizes existing addressing schemes. Otherwise the ATM Forum would need to setup a new, worldwide name registration service, which would be a difficult task to manage and operate. Examples of existing naming schemes are the Internet's DNS domain name hierarchy and X.500's directory information tree hierarchy.

- Dynamic updating of the directory data base

  When an ATM end system address changes, it is important that the ANS database need not be manually reconfigured. The end system should be able to dynamically check and update its address information in the ANS database. This is especially important when the end user does not even necessarily know that an ATM address of the end system has changed. (For example, a manager of the switch could change the address prefix without notifying the users.)

- Security of the dynamic updating process

  Dynamic updating of the ANS database by end systems usually requires that the updates can be authenticated. ANS must therefore include an option to sign the ANS database entries with cryptographic digital signatures. It would also be desirable if ANS could store authenticated public keys in its database in order to make the ANS protocol independent of yet another public key management protocol.

- Automatic configuration of ANS clients

  Finally, automatic configuration of ANS clients is an important requirement. In order to use directory services, the clients need to know one or more ATM addresses of one or more ANS servers. If security is implemented, an ANS client needs a correct public key of at least one zone of the hierarchical name space. A natural solution is that ANS clients learn this information dynamically from an ATM configuration server.

Appendix B: API Interface to ANS

This informative appendix shows an example or a BSD 4.3 compliant application program interface for ANS.

B.1 BSD 4.3 conformant interface

B.1.1 Specifications

The interface to ANS can conform to the BSD 4.3 interface used today to resolve IP protocol family addresses. It consist then of the following calls (defined here in the most common 'C'-syntax)

```
extern int h_errno;

struct hostent *gethostbyname_atmnsap(name)
char *name;

struct hostent *gethostbyname_e164(name)
char *name;

struct hostent *gethostbyaddr(addr, len, type)
char *addr; int len, type;

struct hostent *gethostent()

sethostent(stayopen)
int stayopen;

endhostent()

herror(string)
char *string;
```

where

```
struct    hostent
{
  char  *h_name;        /* official name of host */
  char **h_aliases;     /* alias list */
  int    h_addrtype;    /* host address type */
  int    h_length;      /* length of address */
  char **h_addr_list;   /* list of addresses from nameserver */
};

#define h_addr h_addr_list[0] /* address, for backward compatibility */
```

The members of this structure are:

`h_name` Official name of the host.

`h_aliases` A zero terminated array of alternate names for the host.

`h_addrtype` The type of address being returned

`h_length` The length, in bytes, of the address.

`h_addr_list` A zero terminated array of network addresses for the host. Host addresses are returned in network byte order.

`h_addr` The first address in h_addr_list; this is for backward compatibility.

The `sethostent, endhostent, gethostent,` and `herror` calls are not modified. To support the remaining two routines the semantics of the structure `hostent` has to be extended by the newly supported address types because the component `h_addrtype` today always carries `AF_INET` as answer and the type argument of the `gethostbyaddr` function is supposed to be `AF_INET` in all cases as well. The necessary new specifications are two defines:

```
#define AF_ATMNSAP           3
#define AF_ATME164           8
```

where the values are usually specified in the file `socket.h` and should follow [RFC1700] as proposed in the definitions above. The call `gethostbyname_e164` is supposed to return E.164 addresses and `gethostbyname_atmnsap` ATM-nsap addresses.

The `h_length` is set to the length of the addresses chained in `h_addr_list.`

Additionally, if a socket-like form of address conversion from human readable to internal forms in the API is supported, some new structures and functions may to be provided as existing in today's most BSD 4.3 implementations.

```
struct atmnsap_addr
{
  char atmnsap[20];
}

struct atme164_addr
{
  char atme164[20];
}

struct atmnsap_addr *atmnsap_addr(const char *cp);
char  *atmnsap_ntoa(struct atmnsap_addr in);

struct atme164_addr *atme164_addr(const char *cp);
char  *atme164_ntoa(struct atme164_addr in);
```

The returned and expected human readable strings conform to the form encountered in the master file and described in section 5.4. The returned structures must not be freed by the application and are not thread-safe.

B.1.2 IPng aligned interface

Since introduction of IPv6 included a new addressing scheme, the internet protocol next generation (IPNG) work in progress tries to address the issue of a new application programmer's interface, specifically of the name-resolving calls beside many others. These calls will support address family parameters so functions can be easily implemented to support `AF_ATMNSAP` and `AF_ATME164` as well. Extended functions for address to ASCII conversions and vice versa translation can also be found in this work.

B.1.3 WinSock 2 aligned interface

WinSock 1.1 supported a BSD 4.3 compliant interface for name resolution very much as the one described in section B.1.1, so the functions proposed there can be applied as well. Version 2 of the specification includes the protocol independent name resolution in the current version as an item for further study.