
**NSIF-035-2000**

**(NSIF Document NSIF-DN-9904-043R4)**

**NSIF APPROVED DOCUMENT**

---

**WORK GROUP:** **Distributed Network Management Environment**

---

**TITLE:** **Network Address Resolution and Query Services for CIT/OS Systems**

---

**DATE:** **November 5, 1999**

---

**EDITOR:** **Name: Rémi Theillaud**

**Voice: 33 1 55 91 28 12**

**email: rtheillaud@atos-group.com**

---

**ABSTRACT:**

**This document proposes the use of the LDAP API for fulfilling Network Address Resolution and Query needs for application located on a CIT or an OS system.**

**This document is a superset of and replaces SIF approved document "Directory Services and Network Address Resolution Services for CIT/OS Systems" (SIF-013-1997)**

---

**This document has received the approval of the Network Services and Integration Forum (NSIF).**

# Table of Content



---

**This document has received the approval of the Network Services and Integration Forum (NSIF).**

---

**This document has received the approval of the Network Services and Integration Forum (NSIF).**

# 1 Introduction

## 1.1 Background

This specification complements the SIF specification [SOFT-ARCH] for Management Systems like Craft Interface Terminal (CIT) systems or Operation Systems (OS).

This specification requires the use of the LDAP API (as defined in [RFC1823] and companion standards) on such systems as the standardized API when building applications making a direct use of Network Address Resolution Services, and details profiles for such a purpose.

Use of LDAP on another kind of SONET system (e.g. Network Element) is not precluded, although not part of this specification.

This specification is tied to the T1M1 and SIF documents specifying the use of directory services for Telecommunication Management Networks (TMN), like [T1.245] and [TARP500], as well as [LDAP-SCHEMA].

## 1.2 References

The following documents are referenced in this specification :

| | |
|---|---|
| [SOFT-ARCH] | NSIF-036-2000: CIT/OS Software Platform Architecture |
| [GR-253] | Bellcore GR-253-CORE (1999) : Generic Requirements : SONET Transport Systems : Common Generic Criteria |
| [RFC1278] | IETF RFC 1278 : A string encoding of Presentation Address |
| [RFC1823] | IETF RFC 1823 : The LDAP Application Program Interface |
| [T1.245] | ANSI T1.245 1997 : Directory Services for TMN and SONET |
| [TARP500] | SIF-004-1996 : TARP/500 : the TARP and X.500 Directory Services Interworking Specifications |
| [NARSE] | SIF-013-1997 : Network Address Resolution for CIT/OS system |
| [LDAP-SCHEMA] | SIF-015-1997 : Lightweight Directory Access Protocol: ANSI T1.245 and SIF schema definition |
| [TARP-REQS] | SIF-019-1998 : Interoperability Requirements for TARP |

## 1.3 Acronyms and abbreviations

| | |
|---|---|
| **AET** | Application Entity Title |
| **API** | Application Programming Interface |
| **ASN.1** | Abstract Syntax Notation 1 |
| **CIT** | Craft Interface Terminal |
| **AVA** | Attribute Value Assertion |
| **CMIP** | Common Management Information Protocol |
| **DAP** | (X.500) Directory Access Protocol |
| **DN** | Distinguished Name |
| **DSA** | (X.500) Directory Server Agent |
| **DUA** | (X.500) Directory User Agent |
| **IS/IS L1** | Intermediate System to Intermediate System protocol (Level 1) |
| **LDAP** | Lightweight Directory Access Protocol |
| **NARSE** | Network Address Resolution Service Element |
| **NE** | Network Element |
| **OS** | Operations System |
| **OSI** | Open Systems Interconnection |
| **PKCS** | Public Key Crypto System |
| **RA** | Registration Agent |
| **RDBMS** | Relational Data Base Management System |
| **RDN** | Relative Distinguished Name |
| **RFC** | Request For Comments |
| **RRP** | (ANSI T1.245) Registration Request Protocol |
| **TL1** | Transaction Language 1 |
| **TMN** | Telecommunication Management Network |

---

**This document has received the approval of the Network Services and Integration Forum (NSIF).**

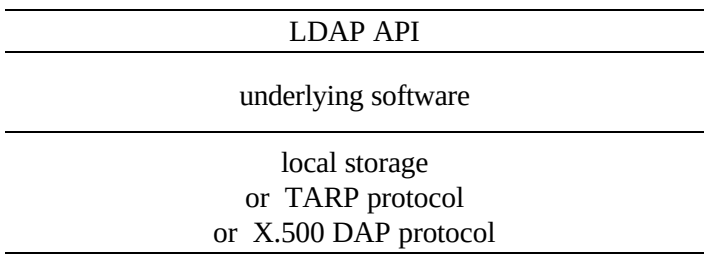
## 1.4 Revision History

| ISSUE | DATE | SUMMARY OF CHANGES |
|---|---|---|
| First Draft | Feb-14-1999 | Proposed for review during Richardson meeting |
| Second Draft | Jun-30-1999 | Update following SIF February meeting |
| Third Draft | Nov-05-1999 | Merge Query ASE contribution with SIF-013-1997 |
| Fourth Draft | Dec-06-1999 | Update following NSIF December meeting in Las Vegas |
| | | |

# 2 Software Architecture and LDAP API

## 2.1 Overview

In order to provide directory services within a Management System (CIT, OS, etc.), the following scheme is defined :




This design allows applications to take advantage of various directory services in a uniform way.

The source of information is expected to be one of the following:

- a directory system fully relying on TARP,
- a distributed or centralized X.500 directory built upon X.500 DSAs conformant to [T1.245] and to the SIF refinements to T1.245 schema (refer to [TARP500] for details),
- a local storage (RDBMS, flat files, etc.) contained in the Management System.

In the X.500 case, it is important to note that actual source of information (as stored in the DSA) may come from many sources (that may be combined) :

- Network Elements or Operation Systems entries populated by a hosted DUA Typically, a co-located RA advertises the DUA of the availability of a DSA.
- « TARP-oriented » entries populated by a TARP500 Gateway (T5GW function, as described in [TARP500]) on behalf of NE/OS implementing only the TARP protocol.
- Manual or automated provisioning by the directory administrator(s).

## 2.2 Software Architecture

Underneath the LDAP API, one or several “mapping modules” will allow to map the requests performed through the API to an actual source of information. Such a source may be either reached through a networking protocol or be directly looked-up in the case of a local database.

Four mapping modules are identified up to now :

a) LDAP API to TARP: directory queries are performed thanks to the local TARP engine;

b) LDAP API to T5: directory queries are performed using either DAP or TARP protocols, following the principles described as the “T5” function in [TARP500];

c) LDAP API to Local Storage: directory queries directly map onto a local database lookup (e.g. RDBMS, flat files, etc.);

d) LDAP API to DAP: directory queries are translated into DAP operations submitted to a DSA. This specification allows the use of the 1988 and 1993 editions of DAP[1].

Note: This document does not propose any mapping of the LDAP API over the LDAP protocol.

It has to be emphasized that the communication between the LDAP API and the mapping modules is a LOCAL matter within the Management System (CIT, OS, etc.). Therefore no assumption is made about using the LDAP protocol or any kind of Inter-Process Communication for such internal communication.

## 2.3 Naming scheme and Information Model

The X.500 schema described in [T1.245] is used in any case. The Target Identifier (TID) resolution is also covered by the information model, following principles described in [TARP500].

The naming tree as described in [T1.245] typically looks like:

| | |
|---|---|
| c=US | Country |
| o=PhoneInc | Organization |
| ou=SouthernDiv | Organization Unit (maybe several levels) |
| cn=Smallville | TMN system (e.g. tmnNE) |
| cn=ftam | application process |
| cn=responder | application entity |

That naming tree is typically separated in two parts:

- geographical/organizational information (ou=z1, ou=z2, o=yy, c=xx)
- TMN systems information (tmnNE entries, and related applicationProcess/applicationEntity entries)

The geographical/organizational information for the local area is described as a "Local Naming Context" (LNC) by [TARP500].

[1] DAP-1993 is an extension of DAP-1988. Therefore, any DSA shall process a valid DAP-1988 query.

That LNC may be either provided automatically through the RRP protocol via the “Name Prefix” field (a Distinguished Name -DN- prefix) to an End System implementing the Registration Agent function described in [T1.245], or locally provisioned.

Another point is that the TARP protocol doesn’t need geographical/organizational information to perform a TID address resolution: it starts by looking within the current IS/IS L1 area, then optionally expands the search to neighbor areas.

The base object whose DN is given in the `ldap_search()` primitive may include either implicit or explicit geographical/organizational information.

### 2.3.1 Explicit geographical information

The explicit form is a complete DN made of the sequence of naming components (RDNs) defining an unambiguous and unique path in the directory tree.

In other words, the explicit form points to the target vertex (or node) of the directory by giving an ordered list of arcs starting from the top object.

Here is an example of an explicit naming form of an applicationEntity object that could represent an FTAM initiator on a NE:

“cn=ftam, cn=Smallville, ou=SouthernDiv, o=PhoneInc, c=US”

Here is another example of an explicit naming form pointing the top object:

“”

### 2.3.2 Implicit geographical information

The implicit form is an incomplete DN made of a sequence of naming components (RDNs) starting at the TMN entry level. That level gathers all sibling entries having a tmnNE object class.

In other words, the implicit form points to the target vertex (or node) of the directory by giving an ordered list of arcs starting from an intermediate node object, called the local naming context. The full DN of the base object specified in the search operation can be constructed by appending the incomplete DN of that base object to the DN of the local naming context.

In T1.245, the DN of the local naming context is also referred as the Name Prefix. In each IS-IS area, a Registration Manager is to distribute that piece of information to the DUAs.

When a LDAP function is called, a fake[2] naming component is used as the end of the DN string when using

[2] This naming component IS NOT included in the directory schema. Therefore, it SHALL NEVER appear on the actual directory queries that are sent out to a remote directory server.

the implicit form:

"npx=."

No other value than "." is currently allowed for that fake naming component.

If one assumes that the Name Prefix is "ou=SouthernDiv, o=PhoneInc, c=US", the FTAM initiator mentioned above can be retrieved by specify an implicit name form:

"cn=ftam, cn=Smallville, npx=."

### 2.3.3 Naming form asymmetry

When a directory application issues a query whose base object has an explicit name form, the provider shall guarantee that it will yield entries using explicit forms.

When a directory application issues a query whose base object has an implicit name form, the provider may select either the implicit or the explicit name form for the returned entries.

It is intended that the explicit form be returned whenever possible. A rationale for that recommendation is the possible need for a directory application to dynamically discover the name prefix by reading the naming attribute of the local naming context object.

### 2.3.4 Protocol mapping:

- when using X.500/DAP, an implicit geographical/organizational information means the local area as described by the Name Prefix given through the RRP protocol. Only TMN systems information needs to be given as a sequence of RDNs in a directory request.
- when using X.500/DAP, an explicit geographical/organizational information means that a DN will have to include the full sequence of RDNs to be used in a directory request.
- when using TARP, the geographical/organizational information is always implicit, meaning that the address resolution will start in the current IS/IS L1 area, then optionally expand to neighbor areas. Only TMN systems information needs to be given as a sequence of RDNs in a directory request.

## 2.4 Why the LDAP API?

The LDAP API is defined in [RFC 1823] and companion standards. This API has the following characteristics:

- it is simple to understand and use (no need to understand ASN.1),
- its external definition is not dependent on operating systems,
- it is easy to find implementations running on UNIX as well as Windows NT,
- its external definition allows to achieve "multi-thread safe" implementations,

- the LDAP protocol and the related API are becoming standards for directory support within the Internet world,
- the LDAP API allows to reach several LDAP directory servers, that may or may not be X.500-based. As an example, some RDBMS mappings exist.

For all these reasons, the LDAP API is required for SONET Management Systems (CIT, OS, etc.) for several purposes:

- Network Address Resolution Service Element (NARSE) : either X.500-based, or TARP-based or local storage-based: needs only a reduced subset of the LDAP API in order to search network address based on application-level criteria.
- General-purpose Database Query Service Element : either X.500-based, or local storage-based. May be easily extended to other kind of sources of information. Sophisticated queries may be performed on the source of information.

# 3 NARSE Profiles

## 3.1 Overview

Applications on a Management System share a common requirement : to be able to perform Network Address Resolution based on some application-level naming scheme.

The intent of NARSE profiles is to specify the mechanisms that are required to fulfill such needs.

This doesn't preclude a software vendor to provide products having additional capabilities, but allows to define a common, minimal subset that can be relied on.

## 3.2 "Core" vs "Extended" Profiles

Network Address Resolution requirements for most applications is limited to translate an application-level name (either a TID or an AET) to an OSI presentation address.

In addition, if only TARP is usable as a directory protocol, the kind of directory requests that are feasible through an LDAP/T5 mapping module are very restricted.

On the other hand, more sophisticated queries might be needed by some applications, using more complex search criteria. But this is only feasible when using T1.245 DAP to access to a true X.500 DSA.

Therefore, two NARSE profiles are defined, allowing two levels of conformance:

1. NARSE "core" profile
2. NARSE "extended" profile

## 3.3 NARSE "Core" profile queries

The Network Address Resolution core subset is aimed at retrieving the necessary information to establish a communication with a managed system by only knowing the logical name of the targeted system.

The logical name can be an AET or a TID. The **implicit naming form shall be used**.

Thus, it allows an application to perform the following query: translation of an incomplete DN or a TID to an OSI presentation address.

The reverse query may also be performed: translation of an OSI presentation address to an application-level name (either a TID or an AET).

All listed mapping are potentially able to render the core part of the Network Address Resolution Service Element.

### 3.3.1 Schema subset

When performing such queries (using **implicit** geographical/organizational information), the base object DN will possibly make use of the following name forms (defined in [T1.245] X.500 schema and its LDAP counterpart [LDAP-SCHEMA]):

- tmnNENameForm
- applProcessNameForm
- applEntityNameForm
- apaeAliasNameForm (dereferencing being typically hidden to the user of the LDAP API)

Therefore only support for these name forms (and corresponding structure rules) is required for that "core" profile.

When performing such queries, only support for applicationEntity object class entries (cf. [T1.245] X.500 schema, and its LDAP counterpart [LDAP-SCHEMA]) retrieval is required. Therefore only support for the corresponding object class, attribute types and syntax is required:

Attributes:

- commonName
- localityName
- presentationAddress (when translating an application-level name to an OSI presentation address, the resulting OSI presentation address will be found in this attribute of the returned applicationEntity entry. The presentation address will be encoded following the principles described in [RFC 1278]).
- organizationName
- supportedApplicationContext

Syntax:

- PrintableString
- PresentationAddress
- OID
- DirectoryString

### 3.3.2 Base objects

An implementation may support only the following base objects:

- a grandson entry of the local naming context (i.e., the base object identifies an applicationEntity entry). One assumes that the RDN of the two arcs have both the same commonName syntax and the same value.

  For instance, "cn=<tid>, cn=<tid>, npx=.". Additional information for using a TID as an application-level name may be found in [TARP500].

- the local naming context, that is "npx=."

Note: The first base object will be typically used for application-level name to OSI presentation address translation. The second one will be typically used for the reverse query.

### 3.3.3 Filters

An implementation shall accept a query without filter for all supported base objects.

An implementation is only required to support the "(objectClass=applicationEntity)" criterion for grandson base objects (i.e., the base object identifies an applicationEntity entry).

An implementation is only required to support the "(& (objectClass=applicationEntity) (presentationAddress=<p-address>))" compound filter in case the base object is the local naming context.

Note: The first filter will be typically used for application-level name to OSI presentation address translation. The second one will be typically used for the reverse query.

### 3.3.4 Scopes

An implementation is allowed to support only `LDAP_SCOPE_BASE` criterion for grandson base objects (i.e., the base object identifies an applicationEntity entry).

An implementation is allowed to support only `LDAP_SCOPE_SUBTREE` when the base object is the local naming context.

Note: The first scope will be typically used for application-level name to OSI presentation address translation. The second one will be typically used for the reverse query.

## 3.4 NARSE "Extended" profile queries

That profile is a superset of the "core" profile: the requests to be achieved are far less restricted. One can refer to Annex I of [T1.245] for the definition of many needs that can be fulfilled with the "extended" profile. For instance, based on that material, the following requests might be issued:

- request the identity of an NE based on its network address;
- request the identity of an NE based on technology, implementation, network, or operation-dependent naming;
- request the identity of NEs based on vendor or locality information, when known;
- request the identity of an NE based on its MIB naming attribute;
- request the value of managedElementId of an NE for use in management protocol exchanges;
- request the identity of an NE based on its function and role;
- request AE-titles of entities with which management associations may be established;
- request the presentation address of application entities based on their AE-titles;
- request the supported application contexts of an application entity;

Queries may be performed using implicit or explicit geographical/organizational information.

All mapping but type a) can be selected.

Beware that a type b) implementation may suffer transient service downgrading if the access to the X.500 server is lost. A clever implementation would automatically back off to a type a) mapping so that it can still deliver the core Network Address Resolution functions.

### 3.4.1 Schema subset

When performing such queries for NARSE purposes, the base object DN will possibly make use of the following name forms (defined in [T1.245] X.500 schema and its LDAP counterpart [LDAP-SCHEMA]):

- countryNameForm
- sOPNameForm
- locNameForm
- orgNameForm
- orgUnitNameForm
- orgUnitLocNameForm
- orgUnitLocOrgNameForm
- tmnNENameForm
- applProcessNameForm
- applEntityNameForm
- apaeAliasNameForm (alias dereferencing being typically hidden to the user of the LDAP API)

Therefore only support for these name forms (and corresponding structure rules) is required for that "extended" profile.

When performing such queries for NARSE purposes, only support for tmnNE or applicationEntity object class entries (cf. [T1.245] X.500 schema, and its LDAP counterpart [LDAP-SCHEMA]) retrieval is required. Therefore only support for the corresponding object class, attribute types and syntax is required for that "extended" profile.

Support for the sdhNEEntry auxiliary object class is not required.

Attributes:

- commonName
- countryName
- localityName
- presentationAddress

- stateOrProvinceName
- organizationName
- supportedApplicationContext
- entityAddress

Syntax:

- PrintableString
- PresentationAddress
- OID
- DirectoryString

### 3.4.2 Base objects

No restriction applies for that profile.

### 3.4.3 Filters

An implementation shall at least support the following matching rules:

- integerMatch
- caseIgnoreMatch
- nodeInfoMatch
- distinguishedNameMatch
- objectIdentifierMatch

An application shall be granted to combine any three rules that are supported by the implementation with a '&' (logical AND) operator.

### 3.4.4 Scope

No restriction applies for that profile: LDAP_SCOPE_BASE, LDAP_SCOPE_SUBTREE and LDAP_ONELEVEL are all allowed.

# 4 Query Service Element

## 4.1 Overview

The Query Service Element is aimed at retrieving any piece of information stored in the TMN directory, as defined by [T1.245] and refined by [LDAP-SCHEMA]. Annex I of [T1245] illustrates many of the retrieval and inventory operations one can think of.

It should be emphasized that NARSE is a subset of the Query SE focusing on the TMN information related to the NEs. These items are scattered in the tmnNE entries and their subtrees. Hence, the Query SE accepts any well-formed NARSE query.

Types c)[3]or d) mappings can be selected.

## 4.2 Schema subset

Any name forms, objects or attributes shown in [LDAPSCHEMA] can be used.

An implementation shall be capable of accepting all valid explicit name form pointing to an entry of the directory, including an empty value.

An implementation shall be capable of accepting all valid implicit name form pointing to an entry of the directory, including a string containing no more than the fake naming component.

## 4.3 Base objects

No restriction applies for that profile.

## 4.4 Filters

An implementation shall at least support the following matching rules:

- integerMatch
- caseIgnoreMatch
- caseIgnoreSubstring
- nodeInfoMatch
- distinguishedNameMatch
- objectIdentifierMatch

[3] It is unlikely that the first mapping type be ever used with Query SE for the cost of maintaining and updating a RDBMS on each OS would soon become prohibitive.

An implementation shall support a filter expression made of ten (10) rules, all combined at the same level with the '&' (AND) operator. Support of multi-level filters is NOT mandated in this profile.

## 4.5 Scope

No restriction applies for that profile.: LDAP_SCOPE_BASE, LDAP_SCOPE_SUBTREE and LDAP_ONELEVEL are all allowed.

# 5 LDAP API Profile

The following section describes a subset of the LDAP API (version 2, as specified in [RFC 1823]), describing the mandatory features to be provided by the LDAP API and the underlying software to provide the services needed for Network Address Resolution.

This description doesn't preclude software vendors from providing more extended features, but allows to specify a subset on which applications may rely in any case.

## 5.1 X.500 schema and LDAP counterpart

[T1.245] standard defines the X.500 schema applicable to the TMN directory for the management of SONET equipment.

[LDAP-SCHEMA] defines the LDAP counterpart to the X.500 directory schema of T1.245. It gives the necessary information for making queries using the LDAP API.

It should be noted that although SIF does not add any object class or attribute to the original T1.245 specification, SIF puts additional constraints on entries standing for NEs having a TARP processor. These constraints do not tear down the general TMN schema but are mandated by [TARP500] in order to achieve internetworking between TARP and the TMN directory.

For NARSE purposes, support for only a subset of the schema is required. See chapters 3.3.1 and 3.4.1 for the required name forms and object classes (and related attributes types and syntax).

## 5.2 LDAP API primitives

### 5.2.1 Synchronous and asynchronous flavors

Both asynchronous and synchronous variants of the set of retained primitives must be implemented.

### 5.2.2 LDAP handle

Direct access to the fields of the LDAP structure is discouraged, except as specified in chapter 5.2.6.2.

### 5.2.3 ldap_open/ldap_open_s

This primitive is required to **locally** connect to the relevant LDAP mapping module.

```
LDAP  *ldap_open (char *hostname, int portno)
LDAP  *ldap_open_s (char *hostname, int portno)
```

The hostname shall be the local host : “localhost”.

The port number is used for distinguishing between various mapping modules. The port number for an LDAP mapping module may be different from the standard LDAP one (389), in order to avoid potential conflicts with other LDAP-based products.

### 5.2.4 ldap_bind/ldap_bind_s

Only the simple flavor is required:

```
ldap_simple_bind (LDAP *ld, char *dn, char *passwd)
ldap_simple_bind_s (LDAP *ld, char *dn, char *passwd)
```

NULL values for `dn` and `passwd` parameters are interpreted as an anonymous bind attempt.

### 5.2.5 ldap_unbind

```
ldap_unbind (LDAP *ld)
```

The synchronous `ldap_unbind()` primitive frees up the directory session. The LDAP API user will never received the responses to outstanding requests it formerly initiated once `ldap_unbind()` has been called. This primitive has no asynchronous match.

An implementation is NOT required to end up the protocol dialogue with its directory server straight away. Though, an implementation should clean up any networking connections due to previous directory traffic after a fixed period of time of idleness.

### 5.2.6 ldap_search/ldap_search_s/ldap_search_st

This primitive is required in order to seek a list of entries meeting the filter criteria or to merely read an entry.

Restrictions apply to the base object, the attributes or the filters depending on the service elements. Chapter 2.3 defines how a base object containing either implicit or explicit geographical/organizational information shall be specified.

#### 5.2.6.1 Scope

Any implementation shall support the three scopes:

- `LDAP_SCOPE_BASE`
- `LDAP_SCOPE_ONELEVEL`
- `LDAP_SCOPE_SUBTREE`

#### 5.2.6.2 Size limit

An implementation may ignore (i.e. assume a value of 0 for) the size limit carried by the `ld_sizelimit` field of the `LDAP` structure unless the size limit is set to one (1), in which case only the first matching entry shall be returned.

#### 5.2.6.3 Deferencing

An implementation should ignore the hint for alias handling carried by the `ld_deref` field of the `LDAP` structure.

For the sake of simplicity, it is recommended that all implementations dereference aliases when searching through the directory as well as when locating/reading an entry.

This algorithm corresponds to the `LDAP_DEREF_ALWAYS` value of the `ld_deref` field.

### 5.2.7 Parsing retrieved entries

All the primitives that help to parse retrieved entries are required:

```
ldap_first_entry(...)
ldap_next_entry(...)
ldap_count_entries(...)
ldap_first_attribute(...)
ldap_next_attribute(...)
ldap_get_values(...)
ldap_get_values_len(...)
ldap_count_values(...)
ldap_count_values_len(...)
ldap_value_free(...)
ldap_value_free_len(...)
ldap_get_dn(...)
ldap_explode_dn(...)
ldap_msgfree(...)
```

### 5.2.8 ldap_result and ldap_result2error

`ldap_result` lets the application obtain the result of a previous operation that was initiated in an asynchronous manner.

If `ldap_result` indicates that an error occurred, then `ldap_result2error` can be invoked to get the error code.

### 5.2.9 ldap_err2string

That primitive is used to translate the error code returned by one of the synchronous LDAP function, or by

function `ldap_result2error`, to a string describing the error.

### 5.2.10 Other primitives

The Query Service and the Network Address Resolution Service require no other primitive than those listed above.

# 6 Future extensions

This section is not part of the specification. It only gives a few hints for future directions.

## 6.1 LDAP v3 and related API

Moving to the version 3 of LDAP and the related upgrade of the LDAP API will have to be studied when the corresponding RFCs will be stable enough.

Thanks to the upward compatibility that should be enforced between the two versions of the LDAP APIs, such a move should be feasible in a smooth way.

## 6.2 Security mechanisms

Four user-security levels have been identified:

- no security at all (anonymous bind)
- user authentication at bind time through name and clear text password
- user authentication at bind time through name and encrypted password
- strong authentication through PKCS mechanisms

This document only requires the use of the first two security levels.

NSIF is currently studying PKCS-based security mechanisms. Once completed, this study will be the basis for the specification of strong authentication in this document.

Note also that appendix B gives a few hints about the use of encrypted password at bind time.

In particular, strong authentication might be studied in order to improve security when establishing associations. That leads to two possible extensions:

- use of an X.500 directory to store public key certificates as well as certificate revocation lists. Such pieces of information might then need to be accessed through the LDAP API and then be typically used for establishing an association using some management protocol (e.g. TL1 or CMIP).
- use of strong authentication during the association establishment with the directory server itself, through the DAP protocol.

# Appendix A: PICS

The following notation is used in that appendix:

| | |
|---|---|
| M | Mandatory |
| O | Optional |
| O.<n> | Optional, but support of at least one of the option labelled O.<n> is required |
| X | Prohibited |
| <Req>: | Applies only when the PICS states that the requirement labelled <Req> is supported. |

## A.1 Service Elements

| No | Description | M/O | Reference |
|---|---|---|---|
| R1 | core of the Network Address Resolution SE | O.1 | |
| R2 | Network Address Resolution SE | O.1 | |
| Q | Query SE | O.1 | |

## A.2 Mapping modules

| No | Description | M/O | Reference |
|---|---|---|---|
| R3 | LDAP API to DAP | R1:O.2 (R2 or Q):O.3 | [T1.245], [TARP500] |
| R4 | LDAP API to TARP | R1:O.2 (R2 or Q):X | [GR-253] |
| R5 | LDAP API to T5 | R1:O.2 (R2 or Q):O.3 | [T1.245] [GR-253] [TARP500] |
| | | | |
| R7 | LDAP API to Local Storage | R1:O.2 (R2 or Q):O.3 | |

## A.3 schemas

| Item | Description | M/O | Reference |
|---|---|---|---|
| CS | Schema for the core NARSE subset | M | |
| NS | NARSE schema | R2:M<br>(R1 or Q):O | |
| QS | Query SE schema | Q:M<br>(R1 or R2):O | [LDAPSCHEMA] |

# Appendix B: User authentication at bind time through encrypted password

This appendix is not part of the specification, but is only intended to provide a few hints for a future extension allowing the use of encrypted password at bind time. Such a specification would be required for security interoperability between applications.

The following ASN.1 definitions have been extracted from X.500 standards:

ISO/IEC 9594-3 / X.511 defines a protected password as:

```
SIGNATURE {OCTET STRING}
```

ISO/IEC 9594-8 / X.509 defines the parameterized type SIGNATURE as:

```
SIGNATURE {OfSignature}    ::= SEQUENCE {
   algorithmIdentifier     AlgorithmIdentifier,
   encrypted          ENCRYPTED { HASHED {OfSignature}}}
```

It also defines the type AlgorithmIdentifier as:

```
AlgorithmIdentifier  ::= SEQUENCE {
   algorithm          ALGORITHM.&id ({SupportedAlgorithms}),
   parameters         ALGORITHM.&id ({SupportedAlgorithms} {@algorithm})
OPTIONAL}
   ALGORITHM          ::= TYPE-IDENTIFIER
```

Hence, NSIF would need to specify:

- Whether applying a hash function to the password is required or not (probably not),
- which algorithm(s) can be used to encrypt the password,

and for each algorithm,

- which OBJECT IDENTIFIER value is used to identify this algorithm,
- optionally, which additional parameters (time stamp, non repeatable value, ...) need to be sent along with this id., to allow the directory server to retrieve the password; more specifically, definition of an ASN.1 type to carry these parameters.

# Appendix C: Implementation Guidelines

## C.1 Mapping over T5

That mapping module allows to process directory requests using either DAP (following T1.245 principles) or TARP protocol. These protocols shall be used following the principles described as the “T5 function” in the [TARP500] document, in order to be able to perform directory requests using one protocol or another.

## C.2 Mapping over TARP

Only requests using **implicit** geographical/organizational information are processed, following TARP principles: try to solve the request in the current IS/IS L1 area, then optionally expand the flooding to adjacent areas.

Since the geographical/organization information is fully unknown on TARP-only system, the Distinguished Names in resulting entries will always use the implicit form.

## C.3 Mapping over DAP

Using the DAP protocol, an X.500 DSA server will be queried that may contain information describing NEs that are T1.245 conformant (i.e. including an X.500 DUA function), but also possibly NEs that are "TARP-oriented" (thanks to the TARP500 gateway mechanism).

Both implicit and explicit geographical/organizational information are allowed. Implicit geographical/organizational information refers to the NamePrefix distributed through the RRP protocol described in T1.245, or locally provisioned.

Since the geographical/organization information is known, the Distinguished Names in resulting entries will always use the explicit form when the resolution has been performed using the DAP protocol.

# Appendix D: C example

```
#include <ldap.h>

   #define EntityAddressMxLength ((int)20)

/*
 * Perform a TID to NSAP translation, using the synchronous flavor of
 *  LDAP API.
 */
static sync_tidToPSap(
   char *cstrTid,
   char *pPSap
   )
{
   LDAP           *ldapHandle;
   int             ldapError;
   LDAPMessage    *pResult;
   LDAPMessage    *pEntry;
   char           *pAttr;
   void           *pPtr;
   struct berval **pValues;

   struct timeval timeout = { (long)20, (long)0 };

   char  cstrBaseObject[128];
   char *cstrAttrArray[] = { "presentationAddress", (char*)0 };

   pPSap[0] = 0;

   /*
    * Connection to the *local* (i.e., hostname is "localhost") LDAP
    *  module;
    * One can choose another port number than the default LDAP one.
    */
   if ( ( ldapHandle = ldap_open( "hostname", LDAP_PORT ) ) == NULL )
   {
```

```
   printf( "ldap_open error\n" );
   return;
}

/*
 * Synchronous bind to the LDAP module; anonymous binding is used here
 *  (no dn, no password).
 */
if ( ( ldapError = ldap_simple_bind_s( ldapHandle, (char*)0, (char*)0 )
     ) != LDAP_SUCCESS
   )
{
   printf( "ldap_simple_bind_s error: %s\n", ldap_err2string(ldapError) );
   return;
}

/*
 * Synchronous search; note that the function ldap_search_s() can also be
 *  called if no timeout needs to be specified.
 * Note: do not use fields ld_sizelimit, ld_timelimit, ld_deref of the
 *  LDAP handle (default values OK as specified in SIF-013-1997), or
 *  possible use of ld_sizelimit (if set to one) as specified in
 *  "Query ASE" document
 */
/*
 * First, build the base object assuming the following directory
 *  structure:
 *                        (local naming context)
 *                                  |
 *                               tmnNE: CN=<TID>
 *                                  |
 *                     (TL1) Application Entity: CN=<TID>
 */
sprintf( cstrBaseObject, "cn=%s, cn=%s, npx=.", cstrTid, cstrTid );

if ( ( ldapError = ldap_search_st( ldapHandle,
                                   cstrBaseObject,
                                   LDAP_SCOPE_BASE,
```

```
                                    "(objectClass=applicationEntity)",
                                    cstrAttrArray,
                                    0, /* Types and values */
                                    &timeout,
                                    &pResult
                                  )
     ) != LDAP_SUCCESS
   )
{
   printf( "ldap_search_st error: %s\n", ldap_err2string(ldapError) );

   ldap_unbind( ldapHandle );
   return;
}

/*
 * Parse the result. Multiple resulting entries are not expected here.
 * A Network address is expected, i.e., binary data, so let's call function
 *  ldap_get_values_len() to retrieve it.
 */
if ( pResult )
{
   if ( ( ldap_count_entries( ldapHandle, pResult ) > 0 )              &&
        ( pEntry = ldap_first_entry( ldapHandle, pResult ) )           &&
        ( pAttr  = ldap_first_attribute( ldapHandle, pEntry, &pPtr ) ) &&
        ( pValues = ldap_get_values_len( ldapHandle, pEntry, pAttr ) ) &&
        ( pValues[0] )
      )
   {
      if ( pValues[0]->bv_len <= EntityAddressMxLength )
      {
         pPSap[0] = pValues[0]->bv_len;
         memcpy( &pPSap[1], pValues[0]->bv_val, pValues[0]->bv_len);
      }
      else
      {
         printf("Result parsing failure: Invalid length!\n");
      }
```

```
         ldap_value_free_len( pValues );
      }
      else
      {
         printf("Result parsing failure!\n");
      }

      /*
       * Free the result
       */
      ldap_msgfree( pResult );
   }

   /*
    * Unbind and close the LDAP connection.
    */
   if ( ( ldapError = ldap_unbind( ldapHandle ) ) != LDAP_SUCCESS )
   {
      printf( "ldap_unbind error: %s\n", ldap_err2string(ldapError) );
      return;
   }

   return;
}

main(
   int     argc,
   char  **argv
   )
{
   char PSap[ EntityAddressMxLength +1 ];
   sync_tidToPSap( "mytid", PSap );
}
```