A Configuration Schema for LDAP Based Directory User Agents
<draft-joslin-config-schema-04.txt>

Status of this Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months. Internet-Drafts may be updated, replaced, or made obsolete by other documents at any time. It is not appropriate to use Internet-Drafts as reference material or to cite them other than as a "working draft" or "work in progress".

To learn the current status of any Internet-Draft, please check the lid-abstracts.txt listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

Distribution of this document is unlimited.

Abstract
This document describes a mechanism for global configuration of similar directory user agents. This document defines a schema for configuration of these DUAs that may be discovered using the Lightweight Directory Access Protocol [RFC2251]. A set of attribute types and an objectclass are proposed, along with specific guidelines for interpreting them. A significant feature of the global configuration policy for DUAs, is a mechanism that allows DUAs to re-configure their schema to that of the end user’s environment. This configuration is achieved through attribute and objectclass mapping. This document is intended to be a skeleton for future documents that describe configuration of specific DUA services.

1. Background & Motivation

The LDAP protocol has brought about a new and nearly ubiquitous acceptance of the directory server. Many new client applications (DUAs) are being created that use LDAP directories for many different services. And although the LDAP protocol has eased the development of these applications, some challenges still exist for both developers and directory administrators.

The authors of this document are implementers of DUAs described by RFC 2307 [14]. In developing these agents, we felt there are several issues that still need to be addressed to ease the deployment and configuration of a large network of these DUAs.

One of these challenges stems from the lack of a utopian schema. A utopian schema would be one that every application developer could agree upon and that would support every application. Unfortunately today, many DUAs define their own schema (like RFC 2307 vs. Microsoft’s Services for Unix [13]) containing similar attributes, but with different attribute names. This can lead to data redundancy within directory entries and give directory administrators unwanted challenges, updating schemas and synchronizing data.

So, one goal of this document is to eliminate data redundancy by having DUAs configure themselves to the schema of the deployed directory, instead of forcing it’s own schema on the directory.

Another goal of this document is to provide the DUA with enough configuration information so that it can discover how to retrieve its data in the directory, such as what locations to search in the directory tree.

Finally, this document intends to describe a configuration method for DUAs that can be shared among many DUAs, on various platforms, providing as such, a configuration profile, the purpose being to
centralize and simplify management of DUAs.

This document is intended to provide the skeleton framework for future drafts, which will describe the individual implementation details for the particular services provided by that DUA. The authors of this document plan to develop such a document for the Network Information Service DUA, described by RFC 2307 or its successor.

We expect that as DUAs take advantage of this configuration scheme, each DUA will require additional configuration parameters, not specified by this document. Thus, we would expect that new auxiliary object classes, containing new configuration attributes will be created, and then joined with the structural class defined by this document to create a configuration profile for a particular DUA service. And that by joining various auxiliary objectclasses for different DUA services, that configuration of various DUA services can be controlled by a single configuration profile entry.

2. General Issues

The schema defined by this document is defined under the "DUA Configuration Schema." This schema is derived from the OID: iso (1) org (3) dod (6) internet (1) private (4) enterprises (1) Hewlett-Packard Company (11) directory (1) LDAP-UX Integration Project (3) DUA Configuration Schema (1). This OID is represented in this document by the keystring "DUAConfSchemaOID" (1.3.6.1.4.1.11.1.3.1).

2.1 Terminology

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 [15].

2.2 Attributes

The attributes and classes defined in this document are summarized below.

The following attributes are defined in this document:

- preferredServerList
- defaultServerList
- defaultSearchBase
- defaultSearchScope
- authenticationMethod
credentialLevel
serviceSearchDescriptor
serviceCredentialLevel
dserviceAuthenticationMethod
attributeMap
objectclassMap
searchTimeLimit
bindTimeLimit
followReferrals
profileTTL

2.3 Object Classes

The following object class is defined in this document:

DUAConfigProfile

2.4 Syntax Definitions

The following syntax definitions are used throughout this document. This document does not define new syntaxes that must be supported by the directory server. The string encoding used by the attributes defined in this document can be found section 5.

keystring as defined by RFC 2252 [2]
descr as defined by RFC 2252 section 4.1
a as defined by RFC 2252 section 4.1
d as defined by RFC 2252 section 4.1
space as defined by RFC 2252 section 4.1
whsp as defined by RFC 2252 section 4.1
base as defined by RFC 2253 [3]
DistinguishedName as defined by RFC 2253 section 2
RelativeDistinguishedName as defined by RFC 2253 section 2
scope as defined by RFC 2255 [5]
IPv4address as defined by RFC 2396 [9]
hostport as defined by RFC 2396 section 3.2.2
port as defined by RFC 2396 section 3.2.2
ipv6reference as defined by RFC 2732 [10]
host as defined by RFC 2732 section 3
serviceID = keystring

3. Attribute Definitions

This section contains attribute definitions to be used by DUAs when discovering their configuration.

( DUAConfSchemaOID.1.0 NAME 'defaultServerList'
DESC 'Default LDAP server host address used by a DUA'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
SINGLE-VALUE )

( DUAConfSchemaOID.1.1 NAME 'defaultSearchBase'
DESC 'Default LDAP base DN used by a DUA'
EQUALITY distinguishedNameMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
SINGLE-VALUE )

( DUAConfSchemaOID.1.2 NAME 'preferredServerList'
DESC 'Preferred LDAP server host addresses to be used by a DUA'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
SINGLE-VALUE )

( DUAConfSchemaOID.1.3 NAME 'searchTimeLimit'
DESC 'Maximum time in seconds a DUA should allow for a search to complete'
EQUALITY integerMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE )

( DUAConfSchemaOID.1.4 NAME 'bindTimeLimit'
DESC 'Maximum time in seconds a DUA should allow for the bind operation to complete'
EQUALITY integerMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE )

( DUAConfSchemaOID.1.5 NAME 'followReferrals'
DESC 'Tells DUA if it should follow referrals returned by a DSA search result'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
SINGLE-VALUE )

( DUAConfSchemaOID.1.6 NAME 'authenticationMethod'
DESC 'A keystring which identifies the type of authentication method used to contact the DSA'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
SINGLE-VALUE )

( DUAConfSchemaOID.1.7 NAME 'profileTTL'
DESC 'Time to live, in seconds, before a client DUA
should re-read this configuration profile'
EQUALITY integerMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE )

( DUAConfSchemaOID.1.14 NAME 'serviceSearchDescriptor'
DESC 'LDAP search descriptor list used by a DUA'
EQUALITY caseExactMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( DUAConfSchemaOID.1.9 NAME 'attributeMap'
DESC 'Attribute mappings used by a DUA'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )

( DUAConfSchemaOID.1.10 NAME 'credentialLevel'
DESC 'Identifies type of credentials a DUA should use when binding to the LDAP server'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
SINGLE-VALUE )

( DUAConfSchemaOID.1.11 NAME 'objectclassMap'
DESC 'Objectclass mappings used by a DUA'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )

( DUAConfSchemaOID.1.12 NAME 'defaultSearchScope'
DESC 'Default search scope used by a DUA'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
SINGLE-VALUE )

( DUAConfSchemaOID.1.13 NAME 'serviceCredentialLevel'
DESC 'Identifies type of credentials a DUA should use when binding to the LDAP server for a specific service'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )

( DUAConfSchemaOID.1.15 NAME 'serviceAuthenticationMethod'
DESC 'Authentication method used by a service of the DUA'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

4. Class Definition
The objectclass below is constructed from the attributes defined in 3, with the exception of the cn attribute, which is defined in RFC 2256 [8]. cn is used to represent the name of the DUA configuration profile.

( DUACnfSchemaOID.2.4 NAME 'DUAConfigProfile'
  SUP top STRUCTURAL
  DESC 'Abstraction of a base configuration for a DUA'
  MUST ( cn )
  MAY ( defaultServerList $ preferredServerList $
         defaultSearchBase $ defaultSearchScope $
         searchTimeLimit $ bindTimeLimit $
         credentialLevel $ authenticationMethod $
         followReferrals $ serviceSearchDescriptor $
         serviceCredentialLevel $ serviceAuthenticationMethod $
         objectclassMap $ attributeMap $
         profileTTL ) )

5. Implementation Details

5.1.1 Interpreting the preferredServerList attribute

Interpretation:

As described by the syntax, the preferredServerList parameter is a white-space separated list of server addresses and associated port numbers. When the DUA needs to contact a DSA, the DUA MUST first attempt to contact one of the servers listed in the preferredServerList attribute. The DUA MUST contact the DSA specified by the first server address in the list. If that DSA is unavailable, the remaining DSAs MUST be queried in the order provided until a connection is established with a DSA. Once a connection with a DSA is established, the DUA SHOULD NOT attempt to establish a connection with the remaining DSAs.

If the DUA is unable to contact any of the DSAs specified by the preferredServerList, the defaultServerList attribute should be examined, as described in 5.1.2. The servers identified by the preferredServerList MUST be contacted before attempting to contact any of the servers specified by the defaultServerList.

Syntax:

serverList       = host *(space [host])
Default Value:

The preferredServerList attribute does not have a default value. Instead a DUA should examine the defaultServerList attribute.

Other attribute notes:

This attribute is used in conjunction with the defaultServerList attribute. Please see section 5.1.2 for additional implementation notes. Determining how the DUA should query the DSAs also depends on the additional configuration attributes, credentialLevel, serviceCredentialLevel, bindTimeLimit, serviceAuthenticationMethod and authenticationMethod. Please review section 5.2 for details on how a Posix DUA should properly bind to a DSA.

Example:

preferredServerList: 1.2.3.4 ldap1.mycorp.com ldap2:1389
[1080::8:800:200C:417A]:1389

5.1.2 Interpreting the defaultServerList attribute

Interpretation:

The defaultServerList attribute MUST only be examined if the preferredServerList attribute is not provided, or the DUA is unable to establish a connection with one of the DSAs specified by the preferredServerList.

If more than one address is provided, the DUA may choose to either accept the order provided, or choose to create its own order, based on what the DUA determines is the "best" order of servers to query. For example, the DUA may choose to examine the server list and choose to query the DSAs in order based on the "closest" server or the server with the least amount of "load." Interpretation of the "best" server order is entirely up to the DUA, and not part of this document.

Once the order of server addresses is determined, the DUA contacts the DSA specified by the first server address in the list. If that DSA is unavailable, the remaining DSAs SHOULD be queried until an available DSA is found or no more DSAs are available. If a server address or port is invalid, the DUA should proceed to the next server address as described just above.
Syntax:

    serverList       = host *(space [host])

Default Value:

    If a defaultServerList attribute is not provided, the DUA should attempt to contact the same DSA that provided the configuration profile entry itself. The default DSA is contacted only if the preferredServerList attribute is also not provided.

Other attribute notes:

    This attribute is used in conjunction with the preferredServerList attribute. Please see section 5.1.1 for additional implementation notes. Determining how the DUA should query the DSAs also depends on the additional configuration attributes, credentialLevel, serviceCredentialLevel, bindTimeLimit, serviceAuthenticationMethod and authenticationMethod. Please review section 5.2 for details on how a DUA should properly contact a DSA.

Example:

    defaultServerList: 1.2.3.4 ldap1.mycorp.com ldap2:1389 [1080::8:800:200C:417A]:1389

5.1.3 Interpreting the defaultSearchBase attribute

Interpretation:

    When a DUA needs to search the DSA for information, this attribute provides the "base" for the search. This parameter can be overridden or appended by the serviceSearchDescriptor attribute. See section 5.1.6.

Syntax:

    Defined by OID 1.3.6.1.4.1.1466.115.121.1.12

Default Value:

    There is no default value for the defaultSearchBase. A DUA MAY define its own method for determining the search base, if the defaultSearchBase is not provided.

Other attribute notes:
This attribute is used in conjunction with the serviceSearchDescriptor attribute. See section 5.1.6.

Example:

defaultSearchBase: dc=mycompany,dc=com

5.1.4 Interpreting the authenticationMethod attribute

Interpretation:

The authenticationMethod attribute defines an ordered list of LDAP bind methods to be used when attempting to contact a DSA. The serviceAuthenticationMethod overrides this value for a particular service (see 5.1.14.) Each method MUST be attempted in the order provided by the attribute, until a successful LDAP bind is performed ("none" is assumed to always be successful). See section 5.2 for more information.

none  - The DUA does not perform an LDAP bind.
simple - The DUA performs an LDAP simple bind.
sasl - The DUA performs an LDAP SASL bind using the specified SASL mechanism and options.
tls  - The DUA performs an LDAP StartTLS operation followed by the specified bind method (for more information refer to section 5.1 of RFC 2830).

Syntax:

authMethod   = method *("," method)
method       = none | simple | sasl | tls
none         = "none"
simple       = "simple"
sasl         = "sasl/" saslmech [ ":" sasloption ]
sasloption   = "auth-conf" | "auth-int"
tls          = "tls:" (none | simple | sasl)
saslmech     = SASL mechanism name as defined in RFC 2222, section 3

Note: Although multiple authentication methods may be specified in the syntax, at most one of each type is allowed.

Default Value:

If the authenticationMethod or serviceAuthenticationMethod (for that particular service) attributes are not provided, the DUA MAY choose to bind to the DSA using any method defined by the DUA. However, if either authenticationMethod or
serviceAuthenticationMethod are provided, the DUA MUST only use the methods specified.

Other attribute notes:

When using TLS, the string "tls:sasl/EXTERNAL" implies that two way authentication is to be performed. Any other TLS authentication method implies one way (DSA side credential) authentication.

Determining how the DUA should bind to the DSAs also depends on the additional configuration attributes, credentialLevel, serviceCredentialLevel, serviceAuthenticationMethod and bindTimeLimit. Please review section 5.2 for details on how to properly bind to a DSA.

Example:

authenticationMethod: tls:simple;sasl/DIGEST-MD5

5.1.5 Interpreting the credentialLevel attribute

Interpretation:

The credentialLevel attribute defines what type(s) of credential(s) the DUA should use when contacting the DSA. The serviceCredentialLevel overrides this value for a particular service (5.1.15.) The credentialLevel can contain more than one credential type, separated by white space.

anonymous - The DUA should not use a credential when binding to the DSA.

proxy - The DUA should use a known proxy identity when binding to the DSA. A proxy identity is a specific credential that was created to represent the DUA. This document does not define how the proxy user should be created, or how the DUA should determine what the proxy user’s credential is. This functionality is up to each implementation.

self - When the DUA is acting on behalf of a "real user" the DUA should attempt to bind to the DSA as that user. The DUA should map the user’s identity to a credential used in the directory.

If the credentialLevel contains more than one credential type, the DUA MUST use the credential types in the order specified. As soon as the DUA is able to successfully bind to the DSA,
the DUA SHOULD NOT attempt to bind using the remaining credential types.

Syntax:

\[
\text{credentialLevel} = \text{level} \ast (\text{space level}) \\
\text{level} = \text{self} | \text{proxy} | \text{anonymous} \\
\text{self} = "self" \\
\text{proxy} = "proxy" \\
\text{anonymous} = "anonymous"
\]

Note: Although multiple credential levels may be specified in the syntax, at most one of each type is allowed. Refer to implementation notes in section 5.2 for additional syntax requirements for the credentialLevel attribute.

Default Value:

If the credentialLevel attribute is not defined, the DUA should not use a credential when binding to the DSA (also known as anonymous.)

Other attribute notes:

Determining how the DUA should bind to the DSAs also depends on the additional configuration attributes, authenticationMethod, serviceAuthenticationMethod, serviceCredentialLevel and bindTimeLimit. Please review section 5.2 for details on how to properly bind to a DSA.

Example:

\[\text{credentialLevel: proxy anonymous}\]

5.1.6 Interpreting the serviceSearchDescriptor attribute

Interpretation:

The serviceSearchDescriptor attribute defines how and where a DUA should search for information for a particular service. The serviceSearchDescriptor contains a serviceID, followed by one or more base-scope-filter triples. These base-scope-filter triples are used to define searches only for the specific service. Multiple base-scope-filters allow the DUA to search for data in multiple locations of the DIT.

In addition to the triples, serviceSearchDescriptor might also contain the DN of an entry that will contain an alternate
profile. The DSA SHOULD re-evaluate the alternate profile and perform searches as specified by that profile.

If the base, as defined in the serviceSearchDescriptor, is followed by the "," (ASCII 0x2C) character, this base is known as a relative base. This relative base may be constructed of one or more RDN components. The DUA MUST define the search base by appending the relative base with the defaultSearchBase.

Syntax:

```
serviceSearchList = serviceID "::" serviceSearchDesc *(";" serviceSearchDesc)
serviceSearchDesc = confReferral | searchDescriptor
searchDescriptor = [base] ["?" [scope] ["?" [filter]]]
confReferral = "ref:" DistinguishedName
base = DistinguishedName | RelativeBaseName
RelativeBaseName = 1*(RelativeDistinguishedName ",")
filter = UTF-8 encoded string
```

If the base or filter contains the ";" (ASCII 0x3B) "?" (ASCII 0x3F) "" (ASCII 0x22) or "\" (ASCII 0x5C) characters, those characters MUST be escaped (preceded with the "\" character.) Alternately the DN may be surrounded by quotes (ASCII 0x22.) Refer to RFC 2253, section 4. If the base or filter are surrounded by quotes, only the "" character needs to be escaped. Any character that is preceded by the "\" character, which does not need to be escaped results in both "\" character and the character itself.

The usage and syntax of the filter string MUST be defined by the DUA service. A suggested syntax would be that as defined by RFC 2254.

If a DUA is performing a search for a particular service which has a serviceSearchDescriptor defined, the DUA MUST set the base, scope and filter as defined. Each base-scope-filter triple represents a single LDAP search operation. If multiple base-scope-filter triples are provided in the serviceSearchDescriptor, the DUA SHOULD perform multiple search requests and in that case it MUST be in the order specified by the serviceSearchDescriptor.

Default Values:

If a serviceSearchDescriptor, or an element their-of, is not
defined for a particular service, the DUA SHOULD create the
base, scope and filter as follows:

  base   - Same as the defaultSearchBase or as
defined by the DUA service.
  scope  - Same as the defaultSearchScope or as
defined by the DUA service.
  filter - Use defaults as defined by DUAs service.

If the defaultSearchBase or defaultSearchScope are not
defined, then the DUA service may use its own default.

Other attribute notes:

If a serviceSearchDescriptor exists for a given service, the
service MUST use at least one base-scope-filter triple in per-
forming searches. It SHOULD perform multiple searches per
service if multiple base-scope-filter triples are defined for
that service.

The details of how the "filter" is interpreted by each DUA’s
service is defined by that service. This means the filter is
NOT REQUIRED to be a legal LDAP filter [4]. Furthermore,
determining how attribute and objectclass mapping affects that
search filter MUST be defined by the service. I.E. The DUA
should specify if the filter has been assumed to already have
been mapped, or if it is expected that mapping would be
applied to the filter. In general practice, implementation
and usability suggests that attribute and objectclass mapping
(sections 5.1.7 and 5.1.12) not be applied to the filter
defined in the serviceSearchDescriptor.

It is assumed the serviceID is unique to a given service
within the scope of any DUA that might use the given profile.

Example:

defaultSearchBase: dc=mycompany,dc=com

serviceSearchDescriptor: email:ou=people,ou=org1,?
one;ou=contractor,?one;
  ref:cn=profile,dc=mycompany,dc=com

In this example, the DUA MUST search in
"ou=people,ou=org1,dc=mycompany,dc=com" first. The DUA then
SHOULD search in "ou=contractor,dc=mycompany,dc=com", and
finally it SHOULD search other locations as specified in the
profile described at "cn=profile,dc=mycompany,dc=com". For more examples, see section 9.

5.1.7 Interpreting the attributeMap attribute

Interpretation:

A DUA SHOULD perform attribute mapping for all LDAP operations performed for a service which has an attributeMap entry. Because attribute mapping is specific to each service within the DUA, a "serviceID" is required as part of the attributeMap syntax. I.E. not all DUA services should necessarily perform the same attribute mapping.

Attribute mapping in general is expected be used to map attributes of similar syntaces as specified by the service supported by the DUA. However, a DUA is NOT REQUIRED to verify syntaxes of mapped attributes. If the DUA does discover that the syntax of the mapped attribute does not match that of the original attribute, the DUA MAY perform translation between the original syntax and the new syntax. When DUAs do support attribute value translation, the list of capabable translations should be documented in a description of the DUA service.

Syntax:

\[
\begin{align*}
\text{attributeMap} & = \text{serviceID":"origAttribute ":" attributes} \\
\text{origAttribute} & = \text{attribute} \\
\text{attributes} & = \text{wattribute} *( \text{space wattribute} ) \\
\text{wattribute} & = \text{whsp newAttribute whsp} \\
\text{newAttribute} & = \text{descr | "*NULL*"} \\
\text{attribute} & = \text{descr}
\end{align*}
\]

Values of the origAttribute are defined by and should be documented for the DUA service, as a list of known supported attributes.

Default Value:

By default, attributes that are used by a DUA service are not mapped unless mapped by the attributeMap attributes. The DUA MUST NOT map an attribute unless it is explicitly defined by an attributeMap attribute.

Other attribute notes:
When an attribute is mapped to the special keystring "*NULL*", the DUA SHOULD NOT request that attribute from the DSA, when performing a search or compare request. If the DUA is also capable of performing modification on the DSA, the DUA SHOULD NOT attempt to modify any attribute which has been mapped to "*NULL*".

It is assumed the serviceID is unique to a given service within the scope of the DSA.

A DUA SHOULD support attribute mapping. If it does, the following additional rules apply:

1) The list of attributes that are allowed to be mapped SHOULD be defined by and documented for the service.

2) Any supported translation of mapping from attributes of dissimilar syntax SHOULD also be defined and documented.

3) If an attribute may be mapped to multiple attributes the DSA SHOULD define a syntax or usage statement for how the new attribute value will be evaluated. Furthermore, the resulting syntax, through translation, of the combined attributes MUST be the same as the attribute being mapped.

4) A DUA MUST support mapping of attributes using the attribute OID. It SHOULD support attribute mapping based on the attribute name.

5) It is recommended that attribute mapping not be applied to parents of the target entries.

6) Attribute mapping is not recursive. In other words, if an attribute has been mapped to a target attribute, that new target attribute MUST NOT be mapped to a third attribute.

7) A given attribute MUST only be mapped once for a given service.

Example:

Suppose a DUA is acting on behalf of an email service. By default the "email" service uses the "mail", "cn" and "sn" attributes to discover mail addresses. However, the email service has been deployed in an environment that uses "employeeName" instead of "cn." And also instead of using the "mail" attribute for email addresses, the "email" attribute is used
for that purpose. In this case, the attribute "cn" can be mapped to "employeeName," allowing the DUA to perform searches using the "employeeName" attribute as part of the search filter, instead of "cn". And "mail" can be mapped to "email" when attempting to retrieve the email address. This mapping is performed by adding the attributeMap attributes to the configuration profile entry as follows (represented in [LDIF]):

attributeMap: email:cn=employeeName
attributeMap: email:mail=email

As described above, the DUA MAY also map a single attribute to multiple attributes. When mapping a single attribute to more than one attribute, the new syntax or usage of the mapped attribute must be intrinsically defined by the DUAs service.

attributeMap: email:cn=firstName lastName

In the above example, the DUA creates the new value by generating space separated string using the values of the mapped attributes. In this case, a special mapping must be defined so that a proper search filter can be created. For further information on this example, please refer to section 9.

Another possibility for multiple attribute mapping might come in when constructing returned attributes. For example, perhaps all email addresses are of a guaranteed syntax of "uid@domain". And in this example, the uid and domain are separate attributes in the directory. The email service may define that if the "mail" attribute is mapped to two different attributes, it will construct the email address as a concatenation of the uid and domain attributes, placing the "@" character between them.

attributeMap: email:mail=uid domain

5.1.8 Interpreting the searchTimeLimit attribute

Interpretation:

The searchTimeLimit attribute defines the maximum time, in seconds, that a DUA should spend performing a search request.

Syntax:

Defined by OID 1.3.6.1.4.1.1466.115.121.1.27.
Default Value:

If the searchTimeLimit attribute is not defined or is zero, the search time limit is not enforced by the DUA.

Other attribute notes:

This time limit only includes the amount of time required to perform the LDAP search operation. If other operations are required, those operations do not need to be considered part of the search time. See bindTimeLimit for the LDAP bind operation.

5.1.9 Interpreting the bindTimeLimit attribute

Interpretation:

The bindTimeLimit attribute defines the maximum time, in seconds, that a DUA should spend performing an LDAP bind request against each server on the preferredServerList or defaultServerList.

Syntax:

Defined by OID 1.3.6.1.4.1.1466.115.121.1.27.

Default Value:

If the bindTimeLimit attribute is not defined or is zero, the bind time limit is not enforced by the DUA.

Other attribute notes:

This time limit only includes the amount of time required to perform the LDAP bind operation. If other operations are required, those operations do not need to be considered part of the bind time. See searchTimeLimit for the LDAP search operation.

5.1.10 Interpreting the followReferrals attribute

Interpretation:

If set to TRUE, the DUA SHOULD follow any referrals if discovered.

If set to FALSE, the DUA MUST NOT follow referrals.
Syntax:

Defined by OID 1.3.6.1.4.1.1466.115.121.1.7.

Default Value:

If the followReferrals attribute is not set or set to an invalid value the default value is TRUE.

5.1.11 Interpreting the profileTTL attribute

Interpretation:

The profileTTL attribute defines how often the DUA SHOULD reload and reconfigure itself using the corresponding configuration profile entry. The value is represented in seconds. Once a DUA reloads the profile entry, it SHOULD re-configure itself with the new values.

Syntax:

Defined by OID 1.3.6.1.4.1.1466.115.121.1.27.

Default Value:

If not specified the DUA MAY use its own reconfiguration policy.

Other attribute notes:

If the profileTTL value is zero, the DUA SHOULD NOT automatically re-load the configuration profile.

5.1.12 Interpreting the objectclassMap attribute

Interpretation:

A DUA MAY perform objectclass mapping for all LDAP operations performed for a service that has an objectclassMap entry. Because objectclass mapping is specific to each service within the DUA, a "serviceID" is required as part of the objectclassMap syntax. I.E. Not all DUA services should necessarily perform the same objectclass mapping.

Objectclass mapping should be used in conjunction with attribute mapping to map the required schema by the service to an equivalent schema that is available in the directory.
Objectclass mapping may or may not be required by a DUA. In general, the objectclass attribute is used primarily in search filters. If a service search descriptor is provided, it is expected that the search filter contains a "correct" search filter (though this is not a requirement,) which does not need to be re-mapped. However, when the service search descriptor is not provided, and the default search filter for that service contains the objectclass attribute, that search filter may be re-defined by objectclass mapping. If a default search filter is not used, it SHOULD be re-defined through the serviceSearchDescriptor. If a serviceSearchDescriptor is defined for a particular service, it SHOULD NOT be re-mapped by either the objectclassMap or attributeMap values.

One condition where the objectclassMap SHOULD be used is when the DUA is providing gateway functionality. In this case, the DUA is acting on behalf of another service, which may pass in a search filter itself. In this type of DUA, the DUA may alter the search filter according to the appropriate attributeMap and objectclassMap values. And in this case, it is also assumed that a serviceSearchDescriptor is not defined.

Syntax:

```
objectclassMap    = serviceID ":" origObjectclass "="
                  objectclass
origObjectclass   = objectclass
objectclass       = keystring
```

Values of the origObjectclass depend on the type of DUA Service using the objectclass mapping feature.

Default Value:

The DUA MUST NOT remap an objectclass unless it is explicitly defined by an objectclassMap attribute.

Other attribute notes:

A DUA SHOULD support objectclass mapping. If it does, the DUA MUST support mapping of objectclasses using the objectclass OID. It SHOULD support objectclass mapping based on the objectclass name.

It is assumed the serviceID is unique to a given service within the scope of the DSA.

Example:
Suppose a DUA is acting on behalf of an email service. By default the "email" service uses the "mail", "cn" and "sn" attributes to discover mail addresses in entries created using inetOrgPerson objectclass [16]. However, the email service has been deployed in an environment that uses entries created using "employee" objectclass. In this case, the attribute "cn" can be mapped to "employeeName", and "inetOrgPerson" can be mapped to "employee", allowing the DUA to perform LDAP operations using the entries that exist in the directory. This mapping is performed by adding attributeMap and objectclassMap attributes to the configuration profile entry as follows (represented in [LDIF]):

attributeMap: email:cn=employeeName
objectclassMap: email:inetOrgPerson=employee

5.1.13 Interpreting the defaultSearchScope attribute

Interpretation:

When a DUA needs to search the DSA for information, this attribute provides the "scope" for the search. This parameter can be overridden by the serviceSearchDescriptor attribute. See section 5.1.6.

Syntax:

scopeSyntax = "base" | "one" | "sub"

Default Value:

The default value for the defaultSearchScope SHOULD be defined by the DUA service. If the default search scope for a service is not defined then the scope should be for the DUA to perform a subtree search.

5.1.14 Interpreting the serviceAuthenticationMethod attribute

Interpretation:

The serviceAuthenticationMethod attribute defines an ordered list of LDAP bind methods to be used when attempting to contact a DSA for a particular service. Interpretation and used of this attribute is the same as 5.1.4, but specific for each service.
Syntax:

```
svAuthMethod    = service ":" method *(";" method)
```

Note: Although multiple authentication methods may be specified in the syntax, at most one of each type is allowed.

Default Value:

If the serviceAuthenticationMethod attribute, or follow its default if not provided.

Other attribute notes:

Determining how the DUA should bind to the DSAs also depends on the additional configuration attributes, credentialLevel, serviceCredentialLevel and bindTimeLimit. Please review section 5.2 for details on how to properly bind to a DSA.

Example:

```
serviceAuthenticationMethod: email:tls:simple;sasl/DIGEST-MD5
```

5.1.15 Interpreting the serviceCredentialLevel attribute

**Interpretation:**

The serviceCredentialLevel attribute defines what type(s) of credential(s) the DUA should use when contacting the DSA for a particular service. Interpretation and used of this attribute are the same as 5.1.5.

**Syntax:**

```
svCredentialLevel = service ":" level *(space level)
```

Refer to implementation notes in section 5.2 for additional syntax requirements for the credentialLevel attribute.

Note: Although multiple credential levels may be specified in the syntax, at most one of each type is allowed.

Default Value:

If the serviceCredentialLevel attribute is not defined, the DUA MUST examine the credentialLevel attribute, or follow its default if not provided.
Other attribute notes:

Determining how the DUA should bind to the DSAs also depends on the additional configuration attributes, serviceAuthenticationMethod, authenticationMethod and bindTimeLimit. Please review section 5.2 for details on how to properly bind to a DSA.

Example:

serviceCredentialLevel: email:proxy anonymous

5.2 Binding to the Directory Server

The DUA SHOULD use the following algorithm when binding to the server:

for (clevel in credLevel) [see note 1]
  if (clevel is "anonymous")
    for (host in hostnames) [see note 2]
      if (server is responding)
        return success
    return failure
  else
    for (amethod in authMethod) [see note 3]
      if (amethod is none)
        for (host in hostnames)
          if (server is responding)
            return success
        return failure
      else
        for (host in hostnames)
          authenticate using amethod and clevel
          if (authentication passed)
            return success
      return failure

Note 1: The credLevel a list of credential levels as defined in serviceCredentialLevel (section 5.1.15) for a given service. If the serviceCredentialLevel is not defined, the DUA MUST examine the credentialLevel attribute.

Note 2: hostnames is the list of servers to contact as defined in 5.1.1 & 5.1.2.

Note 3: The authMethod a list of authentication methods as defined in serviceAuthenticationMethod (section 5.1.14) for a
given service. If the serviceAuthenticationMethod is not defined, the DUA MUST examine the authenticationMethod attribute.

6. Security Considerations

The profile entries MUST be protected against unauthorized modification. Since the profile is most useful if its content is available broadly, it is recommended that the profile entries will be readable anonymously. However, ultimately each service needs to consider implications of providing its service configuration as part of this profile and limit access to the profile entries accordingly. Additionally, the management of the authentication credentials for the DUA is outside the scope of this document and needs to be handled by the DUA.

7. Acknowledgments

There were several additional authors of this document. However we chose to represent only one author per company in the heading. From Sun we also would like to acknowledge Roberto Tam for his design work on Sun’s first LDAP name service product and his input for this document. From Hewlett-Packard we’d like to acknowledge Dave Binder for his work architecting Hewlett-Packard’s LDAP name service product as well as his design guidance on this document. We’d also like to acknowledge Grace Lu from HP, for her input and implementation of HP’s configuration profile manager code.

8. References


9. Examples

In this section we will describe a fictional DUA which provides one service, called the "email" service. This service would be similar to an email client that uses an LDAP directory to discover email addresses based on a textual representation of the recipient's colloquial name.

This email service is defined by default to expect that users with email addresses will be of the "inetOrgPerson" objectclass type [16]. And by default, the "email" service expects the colloquial name to be stored in the "cn" attribute, while it expects the email address to be stored in the "mail" attribute (as one would expect as defined by the inetOrgPerson objectclass.)

As a special feature, the "email" service will perform a special type of attribute mapping, when performing searches. If the "cn" attribute has been mapped to two or more attributes, the "email" service will parse the requested search string and map each white-space separated token into the mapped attributes, respectively.

The default search filter for the "email" service is 
"(objectclass=inetOrgPerson)". The email service also defines that when it performs a name to address discovery, it will wrap the search filter inside a complex search filter as follows:

(&{<filter>}(cn=<name string>))

or if "cn" has been mapped to multiple attributes, that wrapping would appear as follows:

(&{<filter>} {attr1=<token1>}{attr2=<token2}>...)

The below examples show how the "email" service builds it search requests, based on the defined profile. In all cases, the
defaultSearchBase is "o=airius.com" and the defaultSearchScope is undefined.

In addition, for all examples, we assume that the "email" service has been requested to discover the email address for "Jane Hernandez.

Example 1:

serviceSearchDescriptor: email:"ou=marketing,"
base: ou=marketing,o=airius.com
scope: sub
filter: (&(objectclass=inetOrgPerson)(cn~=Jane Hernandez))

Example 2:

serviceSearchDescriptor: email:"ou=marketing,"?one?
(objectclass=inetOrgPerson)(c=us)
attributeMap: email:cn=2.5.4.42 sn

Note: 2.5.4.42 is the OID that represents the "givenName" attribute.

In this example, the email service performs <name string> parsing as described above to generate a complex search filter. The above example results in one search.

base: ou=marketing,o=airius.com
scope: one
filter: (&(&(objectclass=inetOrgPerson)(c=us))
(2.5.4.42~=Jane)(sn~=Hernandez))

Example 3:

serviceSearchDescriptor: email:ou=marketing,"?base
attributeMap: email:cn=name

This example is invalid, because either the quote should have been escaped, or there should have been a leading quote.

Example 4:

serviceSearchDescriptor: email:ou=\mar\keting,\"?base
attributeMap: email:cn=name

base: ou=\mar\keting,"
scope: base
filter (&(objectclass=inetOrgPerson)(name~=Jane Hernandez))

Example 5:

serviceSearchDescriptor: email:ou="marketing",o=supercom

This example is invalid, since the quote was not a leading quote, and thus should have been escaped.

Example 6:

serviceSearchDescriptor: email:??(& (objectclass=person)
 (ou=Org1 \(temporary\)))

base: o=airius.com
scope: sub
filter: ((& (objectclass=person) (ou=Org1 \(Temporary\)))
 (cn~=Jane Henderson)))

Example 7:

serviceSearchDescriptor: email:"ou=funny?org,"

base: ou=funny?org,o=airius.com
scope: sub
filter (&(objectclass/inetOrgPerson)(cn~=Jane Hernandez))

10. Author’s Addresses

Luke Howard
PADL Software Pty. Ltd.
PO Box 59
Central Park Vic 3145
Australia
EMail: lukeh@padl.com

Bob Joslin
Hewlett-Packard Company
19420 Homestead RD MS43-LF
Cupertino, CA 95014
USA
Phone: +1 408 447-3044
EMail: bob_joslin@hp.com