

Network Working Group
Request for Comments: 4876
Category: Informational

B. Neal-Joslin, Ed.
HP
L. Howard
PADL
M. Ansari
Infoblox
May 2007

A Configuration Profile Schema for Lightweight Directory Access Protocol (LDAP)-Based Agents

Status of This Memo

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

Copyright Notice

Copyright (C) The IETF Trust (2007).

IESG Note

This RFC is not a candidate for any level of Internet Standard. The IETF disclaims any knowledge of the fitness of this RFC for any purpose and in particular notes that the decision to publish is not based on IETF review for such things as security, congestion control, or inappropriate interaction with deployed protocols. The RFC Editor has chosen to publish this document at its discretion. Readers of this document should exercise caution in evaluating its value for implementation and deployment. See RFC 3932 for more information.

Abstract

This document consists of two primary components, a schema for agents that make use of the Lightweight Directory Access protocol (LDAP) and a proposed use case of that schema, for distributed configuration of similar directory user agents. A set of attribute types and an object class are proposed. In the proposed use case, directory user agents (DUAs) can use this schema to determine directory data location and access parameters for specific services they support. In addition, in the proposed use case, attribute and object class mapping allows DUAs to reconfigure their expected (default) schema to match that of the end user's environment. This document is intended to be a skeleton for future documents that describe configuration of specific DUA services.

Table of Contents

1. Background and Motivation	3
2. General Information	4
2.1. Requirements Notation	4
2.2. Attributes Summary	5
2.3. Object Classes Summary	5
2.4. Common Syntax/Encoding Definitions	5
3. Schema Definition	6
3.1. Attribute Definitions	6
3.2. Class Definition	9
4. DUA Implementation Details	10
4.1. Interpreting the preferredServerList Attribute	10
4.2. Interpreting the defaultServerList Attribute	11
4.3. Interpreting the defaultSearchBase Attribute	12
4.4. Interpreting the authenticationMethod Attribute	13
4.5. Interpreting the credentialLevel Attribute	15
4.6. Interpreting the serviceSearchDescriptor Attribute	16
4.7. Interpreting the attributeMap Attribute	20
4.8. Interpreting the searchTimeLimit Attribute	23
4.9. Interpreting the bindTimeLimit Attribute	23
4.10. Interpreting the followReferrals Attribute	24
4.11. Interpreting the dereferenceAliases Attribute	24
4.12. Interpreting the profileTTL Attribute	24
4.13. Interpreting the objectclassMap Attribute	25
4.14. Interpreting the defaultSearchScope Attribute	27
4.15. Interpreting the serviceAuthenticationMethod Attribute	27
4.16. Interpreting the serviceCredentialLevel Attribute	28
5. Binding to the Directory Server	29
6. Security Considerations	29
7. Acknowledgments	30
8. IANA Considerations	30
8.1. Registration of Object Classes	31
8.2. Registration of Attribute Types	31
9. References	33
9.1. Normative References	33
9.2. Informative References	34
Appendix A. Examples	35

1. Background and Motivation

LDAP [RFC4510] has brought about a nearly ubiquitous acceptance of the directory server. Many 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 [RFC2307]. In developing these agents, we felt there were 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, even when they provide similar services (like RFC 2307 vs. Microsoft's Services for Unix [MSSFU]). These schemas contain similar attributes, but use different attribute names. This can lead to data redundancy within directory entries and cause directory administrators unwanted challenges, updating schemas and synchronizing data. Or, in a more common case, two or more applications may agree on common schema elements, but choose a different schema for other elements of data that might also be shareable between the applications. While data synchronization and translation tools exist, the authors of this document believe there is value in providing this capability in the directory user agent itself.

Aside from proposing a schema for general use, 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 the DUA'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 of this profile is to centralize and simplify management of DUAs.

This document is intended to provide the skeleton framework for future documents that 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 that contain 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. By joining various auxiliary object classes for different DUA services, the configuration of various DUA services can be controlled by a single configuration profile entry.

2. General Information

The schema defined by this document is defined under the "DUA Configuration Schema". This schema is derived from the object identifier (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 keystore "DUACnfSchemaOID" (1.3.6.1.4.1.11.1.3.1).

2.1. Requirements Notation

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

2.2. Attributes Summary

The following attributes are defined in this document:

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

2.3. Object Classes Summary

The following object class is defined in this document:

- DUAConfigProfile

2.4. Common Syntax/Encoding Definitions

The proposed string encodings used by the attributes defined in this document can be found in Section 4. This document makes use of ABNF [RFC4234] for defining new encodings.

The following syntax definitions are used throughout this document.

The list of used syntaxes are:

Key	Source
keystring	as defined by [RFC4512] Section 1.4
descr	as defined by [RFC4512] Section 1.4
SP	as defined by [RFC4512] Section 1.4
WSP	as defined by [RFC4512] Section 1.4
base	as defined by distinguishedName in [RFC4514]
distinguishedName	as defined by [RFC4514] Section 2
relativeDistinguishedName	as defined by [RFC4514] Section 2
scope	as defined by [RFC4516] Section 2
host	as defined by [RFC3986] Section 3.2.2
hostport	host [":" port]
port	as defined by [RFC3986] Section 3.2.3
serviceID	same as keystring

This document does not define new syntaxes that must be supported by the directory server. Instead, these syntaxes are merely expected to be interpreted by the DUA. As referenced in the schema definition in Section 3, most encodings are expected to be stored in attributes using common syntaxes, such as the Directory String syntax, as defined in Section 3.3.6 of [RFC4517]. Refer to RFC 4517 for additional syntaxes used by this schema.

3. Schema Definition

This section defines a proposed schema. This schema does not require definition of new matching rules or syntaxes, and it may be used for any purpose seen. A proposed use of this schema to support elements of configuration of a directory user agent is described in Section 4.

3.1. Attribute Definitions

This section contains attribute definitions used by agents. The syntax used to describe these attributes is defined in [RFC4512], Section 4.1.2. Individual syntaxes and matching rules used within these descriptions are described in [RFC4517], Sections 3.3 and 4.2, respectively.

```
( 1.3.6.1.4.1.11.1.3.1.1.0 NAME 'defaultServerList'
  DESC 'List of default servers'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE )

( 1.3.6.1.4.1.11.1.3.1.1.1 NAME 'defaultSearchBase'
  DESC 'Default base for searches'
  EQUALITY distinguishedNameMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
  SINGLE-VALUE )

( 1.3.6.1.4.1.11.1.3.1.1.2 NAME 'preferredServerList'
  DESC 'List of preferred servers'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE )

( 1.3.6.1.4.1.11.1.3.1.1.3 NAME 'searchTimeLimit'
  DESC 'Maximum time an agent or service allows for a
  search to complete'
  EQUALITY integerMatch
  ORDERING integerOrderingMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )

( 1.3.6.1.4.1.11.1.3.1.1.4 NAME 'bindTimeLimit'
  DESC 'Maximum time an agent or service allows for a
  bind operation to complete'
  EQUALITY integerMatch
  ORDERING integerOrderingMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )

( 1.3.6.1.4.1.11.1.3.1.1.5 NAME 'followReferrals'
  DESC 'An agent or service does or should follow referrals'
  EQUALITY booleanMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
  SINGLE-VALUE )
```

- (1.3.6.1.4.1.11.1.3.1.1.6 NAME 'authenticationMethod'
DESC 'Identifies the types of authentication methods either
used, required, or provided by a service or peer'
EQUALITY caseIgnoreMatch
SUBSTR caseIgnoreSubstringsMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
SINGLE-VALUE)
- (1.3.6.1.4.1.11.1.3.1.1.7 NAME 'profileTTL'
DESC 'Time to live, in seconds, before a profile is
considered stale'
EQUALITY integerMatch
ORDERING integerOrderingMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)
- (1.3.6.1.4.1.11.1.3.1.1.9 NAME 'attributeMap'
DESC 'Attribute mappings used, required, or supported by an
agent or service'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26)
- (1.3.6.1.4.1.11.1.3.1.1.10 NAME 'credentialLevel'
DESC 'Identifies type of credentials either used, required,
or supported by an agent or service'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
SINGLE-VALUE)
- (1.3.6.1.4.1.11.1.3.1.1.11 NAME 'objectclassMap'
DESC 'Object class mappings used, required, or supported by
an agent or service'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26)
- (1.3.6.1.4.1.11.1.3.1.1.12 NAME 'defaultSearchScope'
DESC 'Default scope used when performing a search'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
SINGLE-VALUE)


```
( 1.3.6.1.4.1.11.1.3.1.1.13 NAME 'serviceCredentialLevel'
  DESC 'Specifies the type of credentials either used, required,
  or supported by a specific service'
  EQUALITY caseIgnoreIA5Match
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )

( 1.3.6.1.4.1.11.1.3.1.1.14 NAME 'serviceSearchDescriptor'
  DESC 'Specifies search descriptors required, used, or
  supported by a particular service or agent'
  EQUALITY caseExactMatch
  SUBSTR caseExactSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.11.1.3.1.1.15 NAME 'serviceAuthenticationMethod'
  DESC 'Specifies types authentication methods either
  used, required, or supported by a particular service'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.11.1.3.1.1.16 NAME 'dereferenceAliases'
  DESC 'Specifies if a service or agent either requires,
  supports, or uses dereferencing of aliases.'
  EQUALITY booleanMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
  SINGLE-VALUE )
```

3.2. Class Definition

The object class below is constructed from the attributes defined in Section 3.1, with the exception of the "cn" attribute, which is defined in [RFC4519]. "cn" is used to represent the name of the DUA configuration profile and is recommended for the relative distinguished name (RDN) [RFC4514] naming attribute. This object class is used specifically by the DUA described in Section 4. The syntax used to describe this object class is defined in [RFC4512], Section 4.1.1.

```
( 1.3.6.1.4.1.11.1.3.1.2.5 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 $ dereferenceAliases $
        serviceSearchDescriptor $ serviceCredentialLevel $
        serviceAuthenticationMethod $ objectclassMap $
        attributeMap $ profileTTL ) )
```

4. DUA Implementation Details

This section describes an implementation of the schema described in Section 3. Details about how a DUA should format and interpret the defined attributes are described below. Agents that make use of the DUAConfigProfile object class are expected to follow the specifications in this section.

Note: Many of the subsections below contain examples. Unless otherwise specified, these examples are rendered using the LDAP Data Interchange Format (LDIF) [RFC2849].

4.1. Interpreting the preferredServerList Attribute

Interpretation:

As described by the syntax, the preferredServerList parameter is a whitespace-separated list of server addresses and associated port numbers. When the DUA needs to contact a directory server agent (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 (left to right) 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. The purpose of enumerating multiple DSAs is not for supplemental data, but for high availability of replicated data. This is also the main reason why an LDAP URL [RFC3986] syntax was not selected for this document.

If the DUA is unable to contact any of the DSAs specified by the preferredServerList, the defaultServerList attribute MUST be examined, as described in Section 4.2. The servers identified by

the preferredServerList MUST be contacted before attempting to contact any of the servers specified by the defaultServerList.

Syntax:

serverList = hostport *(SP [hostport])

Default Value:

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

Other attribute notes:

This attribute is used in conjunction with the defaultServerList attribute. Please see Section 4.2 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 for details on how a DUA should properly bind to a DSA.

Example:

```
preferredServerList: 192.168.169.170 ldap1.mycorp.com  
ldap2:1389 [1080::8:800:200C:417A]:389
```

4.2. 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 any of the DSAs specified by the preferredServerList.

If more than one address is provided, the DUA may choose either to accept the order provided or to create its own order, based on what the DUA determines is the "best" order of DSAs to query. For example, the DUA may choose to examine the server list and 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 = hostport *(SP [hostport])
```

Default Value:

If a defaultServerList attribute is not provided, the DUA MAY 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 4.1 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 for details on how a DUA should properly contact a DSA.

Example:

```
defaultServerList: 192.168.169.170 ldap1.mycorp.com  
                  ldap2:1389 [1080::8:800:200C:417A]:5912
```

4.3. 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 4.6.

Syntax:

Defined by OID 1.3.6.1.4.1.1466.115.121.1.12 [RFC4517].

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 4.6.

Example:

```
defaultSearchBase: dc=mycompany,dc=com
```

4.4. 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 Section 4.15). 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). However, the DUA MAY skip over one or more methods. See Section 5 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 Simple Authentication and Security Layer (SASL) [RFC4422] 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 4.14 of [RFC4511]).

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 [SASLMECH]
```

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

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 is provided, the DUA MUST only use the methods specified.

Other attribute notes:

When using TLS, the string "tls:sasl/EXTERNAL" implies that both client and server (DSA and DUA) authentications are to be performed. Any other TLS authentication method implies server-only (DSA side credential) authentication, along with the other SASL method used for DUA-side 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 for details on how to properly bind to a DSA.

Example:

```
authenticationMethod: tls:simple;sasl/DIGEST-MD5
```

```
(see [RFC2831])
```

4.5. Interpreting the credentialLevel Attribute

Interpretation:

The credentialLevel attribute defines what type(s) of credential(s) the DUA MUST use when contacting the DSA. The serviceCredentialLevel overrides this value for a particular service (Section 4.16). The credentialLevel can contain more than one credential type, separated by whitespace.

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 known identity, the DUA MUST attempt to bind to the DSA as that identity. The DUA should contain methods to determine the identity of the user such that the identity can be authenticated by the directory server using the defined authentication methods.

If the credentialLevel contains more than one credential type, the DUA MUST use the credential types in the order specified. However, the DUA MAY skip over one or more credential types. 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:

```
credentialLevel    = level *(SP level)

level              = self / proxy / anonymous

self              = "self"

proxy             = "proxy"

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 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 for details on how to properly bind to a DSA.

Example:

```
credentialLevel: proxy anonymous
```

4.6. 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 in the directory information tree (DIT). Although this syntax is very similar to the LDAP URL [RFC3986], this document requires the ability to supply multiple hosts as

part of the configuration of the DSA. In addition, an ordered list of search descriptors is required, which cannot be specified by the LDAP URL.

The 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. In this case, 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] ["?" [scopeSyntax] ["?" [filter]]]

confReferral      = "ref:" distinguishedName

base              = distinguishedName / relativeBaseName

relativeBaseName  = 1*(relativeDistinguishedName ",")

filter            = UTF-8 encoded string

```

If the confReferral, base, relativeBaseName, or filter contains the ";" (ASCII 0x3B), "?" (ASCII 0x3F), "" (ASCII 0x22), or "\" (ASCII 0x5C) characters, those characters MUST be escaped (preceded by the "\" character). Alternately, the DN may be surrounded by quotes (ASCII 0x22). Refer to RFC 4514. If the confReferral, base, relativeBaseName, or filter are surrounded by quotes, only the "" character needs to be escaped. Any character that does not need to be escaped, and yet is preceded by the "\" character, results in both the "\" 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 defined by [RFC4515].

If a DUA is performing a search for a particular service that 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`.

FYI: Service search descriptors do not exactly follow the LDAP URL syntax [RFC4516]. The reasoning for this difference is to separate the host name(s) from the filter. This allows the DUA to have a more flexible solution in choosing its DSA.

Default Value:

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

base - Same as the `defaultSearchBase`.

scope - Same as the `defaultSearchScope`.

filter - Use defaults as defined by DUA's service.

If the `defaultSearchBase` or `defaultSearchScope` is 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 performing 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 [RFC4515]. Furthermore, determining how attribute and object class mapping affects that search filter MUST be defined by the service. That is, the DUA SHOULD specify if the attributes in the filter are assumed to already have been mapped, or if it is expected that attribute mapping (see Section 4.7) would be applied to the filter. In general practice, implementation and usability suggests that attribute and object class mapping (Sections 4.7 and 4.13) SHOULD NOT be applied to the filter defined in the `serviceSearchDescriptor`.

The serviceID is unique to a given service within the scope of any DUA that might use the given profile, and should be defined by that service. Registration of serviceIDs is not addressed by this document. However, as per the guidance at the end of Section 1, when DUA developers define their use of the DUAConfigProfile schema, they will define the serviceIDs used by that DUA.

searchGuide and enhancedSearchGuide [RFC4517]:

There are a few reasons why the authors chose not to take advantage of the existing searchGuide and enhancedSearchGuide attributes and related syntaxes. While the enhancedSearchGuide met a number of the serviceSearchDescriptor requirements, serviceSearchDescriptor was developed primarily to support associating search operations with services. Multiple services could be configured using the same profile, thus requiring the serviceID to be specified together with the search descriptor information. A few other reasons for not using enhancedSearchGuide include:

- The need to specify alternate search bases, including the ability to specify search bases that are relative to the parent defaultSearchBase.

- The need to specify alternate profiles using the "ref:" syntax.

- The ability for individual services to specify their own syntaxes for the format of the search filter.

- The authors' belief that the user community is more familiar with the search filter syntax described by RFC 4515 than with that described by the enhancedSearchGuide syntax.

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 Appendix A.

4.7. Interpreting the attributeMap Attribute

Interpretation:

A DUA SHOULD perform attribute mapping for all LDAP operations performed for a service that 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. That is, not all DUA services should necessarily perform the same attribute mapping.

Attribute mapping in general is expected to be used to map attributes of similar syntaxes 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 method and list of capable translations SHOULD be documented in a description of the DUA service.

Syntax:

```
attributeMap      = serviceID ":" origAttribute "=" attributes
origAttribute     = attribute
attributes        = wattribute *( SP wattribute )
wattribute        = WSP newAttribute WSP
newAttribute      = descr / "*NULL*"
attribute         = descr
```

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 SHOULD NOT map an attribute unless it is explicitly defined by an attributeMap attribute.

Other attribute notes:

When an attribute is mapped to the special kestring `"*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 constructed. Furthermore, the resulting translated syntax 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". Also, instead of using the "mail" attribute for email addresses, the "email" attribute is used. 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". Also, "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 [RFC2849]):

```
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 a 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 Appendix A.

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". 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 two attributes (uid and domain), placing the "@" character between them.

```
attributeMap: email:mail=uid domain
```

Note: The attributeMap attribute contains only a list of attribute names that should be mapped, not the definition of how syntax translation should be performed. The process used to perform attribute value syntax translation (such as translating a uid to a DN) and/or joining of multiple attribute values to form the target syntax (such as in the above email example) is up to the service. The attribute list defined in the attributeMap merely provides the attributes that would be used as inputs to the translation function provided by the service.

4.8. Interpreting the searchTimeLimit Attribute

Interpretation:

The searchTimeLimit attribute defines the maximum time, in seconds, that the DUA SHOULD allow for a search request to complete.

Syntax:

Defined by OID 1.3.6.1.4.1.1466.115.121.1.27 [RFC4517].

Default Value:

If the searchTimeLimit attribute is not defined or is zero, the searchTimeLimit SHOULD NOT be 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, they do not need to be considered part of the search time. See bindTimeLimit for the LDAP bind operation.

4.9. Interpreting the bindTimeLimit Attribute

Interpretation:

The bindTimeLimit attribute defines the maximum time, in seconds, that a DUA SHOULD allow for the bind request to complete when performed 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 bindTimeLimit SHOULD NOT be 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.

4.10. 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 [RFC4517].

Default Value:

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

4.11. Interpreting the dereferenceAliases Attribute

Interpretation:

If set to TRUE, the DUA SHOULD enable alias dereferencing.

If set to FALSE, the DUA MUST NOT enable alias dereferencing.

Syntax:

Defined by OID 1.3.6.1.4.1.1466.115.121.1.7.

Default Value:

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

4.12. 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 reconfigure 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 reload the configuration profile.

4.13. Interpreting the objectclassMap Attribute

Interpretation:

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

Object class mapping SHOULD be used in conjunction with attribute mapping to map the schema required by the service to an equivalent schema that is available in the directory.

Object class mapping may or may not be required by a DUA. Often, the objectclass attribute is used in search filters. Section 4.7 recommends that attribute mapping not be applied to the serviceSearchDescriptor. Thus, if the default object classes are not used in a DUA deployment, typically only the serviceSearchDescriptor needs to be defined to reflect that mapping. However, when the service search descriptor is not provided, and the default search filter for that service contains the objectclass attribute, that search filter SHOULD be redefined by object class mapping, if defined. If a default search filter is not used, it SHOULD be redefined through the serviceSearchDescriptor. If a serviceSearchDescriptor is defined for a particular service, it SHOULD NOT be remapped 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. In this case, it is also assumed that a serviceSearchDescriptor is not defined.

Syntax:

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

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

Default Value:

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

Other attribute notes:

A DUA SHOULD support object class mapping. If it does, the DUA MUST support mapping of object classes using the objectclass OID. It SHOULD support object class mapping based on the object class 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 object class [RFC2789]. However, the email service has been deployed in an environment that uses entries created using "employee" object class. 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 [RFC2849]):

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

4.14. 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 4.6.

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.

4.15. 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 use of this attribute is the same as Section 4.4, but specific for each service.

Syntax:

svAuthMethod = serviceID ":" 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 is not provided, the authenticationMethod SHOULD be followed, or its default.

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 for details on how to properly bind to a DSA.

Example:

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

4.16. 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 use of this attribute are the same as Section 4.5.

Syntax:

```
svCredentialLevel = serviceID ":" level *(SP level)
```

Refer to implementation notes in Section 5 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 if one is not provided, the DUA must follow its default.

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 for details on how to properly bind to a DSA.

Example:

```
serviceCredentialLevel: email:proxy anonymous
```

5. 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
return failure
```

Note 1: The credLevel is a list of credential levels as defined in serviceCredentialLevel (Section 4.16) 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 Sections 4.1 and 4.2.

Note 3: The authMethod is a list of authentication methods as defined in serviceAuthenticationMethod (Section 4.15) 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. Each service needs to consider implications of providing its service configuration as part of this profile and limit access to the profile entries accordingly.

The management of the authentication credentials for the DUA is outside the scope of this document and needs to be handled by the DUA.

Since the DUA needs to know how to properly bind to the directory server, the access control configuration of the DSA MUST assure that the DSA can view all the elements of the DUAConfigProfile attributes. For example, if the credentialLevel attribute contains "Self", but the DSA is unable to access the credentialLevel attribute, the DUA will instead attempt an anonymous connection to the directory server.

The algorithm described by Section 5 also has security considerations. Altering that design will alter the security aspects of the configuration profile.

At times, DUAs connect to multiple directory servers in order to support potential high-availability and/or performance requirements. As such, each directory server specified in the preferredServer list and defaultServerList MUST contain the same (replicated) data and be part of the same security domain. This means the directory-supported authentication methods, authentication policies, and access control policies for directory data are exactly the same across all the defined directory servers.

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 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. IANA Considerations

This document defines new LDAP attributes and an object class for object identifier descriptors. As specified by Section 3.4 and required by Section 4 of [RFC4520], this document registers new descriptors as follows per the Expert Review.

8.1. Registration of Object Classes

Subject: Request for LDAP Descriptor Registration

Descriptor (short name): DUAConfigProfile

Object Identifier: 1.3.6.1.4.1.11.1.3.1.2.5

Person & email address to contact for further information:
See "Author/Change Controller"

Usage: object class

Specification: RFC 4876

Author/Change Controller:

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

Comments:

See also the associated request for the defaultServerList, defaultSearchBase, preferredServerList, searchTimeLimit, bindTimeLimit, followReferrals, authenticationMethod, profileTTL, attributeMap, credentialLevel, objectclassMap, defaultSearchScope, serviceCredentialLevel, serviceSearchDescriptor, serviceAuthenticationMethod, and dereferenceAliases attribute types.

8.2. Registration of Attribute Types

Subject: Request for LDAP Descriptor Registration

Descriptor (short name): See comments

Object Identifier: See comments

Person & email address to contact for further information:
See "Author/Change Controller"

Usage: attribute type

Specification: RFC 4876

Author/Change Controller:

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

Comments:

The following object identifiers and associated attribute types have been registered.

OID	Attribute Type
1.3.6.1.4.1.11.1.3.1.1.0	defaultServerList
1.3.6.1.4.1.11.1.3.1.1.1	defaultSearchBase
1.3.6.1.4.1.11.1.3.1.1.2	preferredServerList
1.3.6.1.4.1.11.1.3.1.1.3	searchTimeLimit
1.3.6.1.4.1.11.1.3.1.1.4	bindTimeLimit
1.3.6.1.4.1.11.1.3.1.1.5	followReferrals
1.3.6.1.4.1.11.1.3.1.1.6	authenticationMethod
1.3.6.1.4.1.11.1.3.1.1.7	profileTTL
1.3.6.1.4.1.11.1.3.1.1.9	attributeMap
1.3.6.1.4.1.11.1.3.1.1.10	credentialLevel
1.3.6.1.4.1.11.1.3.1.1.11	objectclassMap
1.3.6.1.4.1.11.1.3.1.1.12	defaultSearchScope
1.3.6.1.4.1.11.1.3.1.1.13	serviceCredentialLevel
1.3.6.1.4.1.11.1.3.1.1.14	serviceSearchDescriptor
1.3.6.1.4.1.11.1.3.1.1.15	serviceAuthenticationMethod
1.3.6.1.4.1.11.1.3.1.1.16	dereferenceAliases

Please also see the associated registration request for the DUACfgProfile object class.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.
- [RFC4510] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.
- [RFC4511] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.
- [RFC4512] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Directory Information Models", RFC 4512, June 2006.
- [RFC4514] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, June 2006.
- [RFC4516] Smith, M. and T. Howes, "Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator", RFC 4516, June 2006.
- [RFC4517] Legg, S., "Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules", RFC 4517, June 2006.
- [RFC4519] Sciberras, A., "Lightweight Directory Access Protocol (LDAP): Schema for User Applications", RFC 4519, June 2006.
- [SASLMECH] IANA, "SIMPLE AUTHENTICATION AND SECURITY LAYER (SASL) MECHANISMS", July 2006, <<http://www.iana.org/assignments/sasl-mechanisms>>.

9.2. Informative References

- [MSSFU] Microsoft Corporation, "Windows Services for Unix 3.5", <<http://www.microsoft.com/windows/sfu/>>.
- [RFC2307] Howard, L., "An Approach for Using LDAP as a Network Information Service", RFC 2307, March 1998.
- [RFC2789] Freed, N. and S. Kille, "Mail Monitoring MIB", RFC 2789, March 2000.
- [RFC2831] Leach, P. and C. Newman, "Using Digest Authentication as a SASL Mechanism", RFC 2831, May 2000.
- [RFC2849] Good, G., "The LDAP Data Interchange Format (LDIF) - Technical Specification", RFC 2849, June 2000.
- [RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.
- [RFC4515] Smith, M. and T. Howes, "Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters", RFC 4515, June 2006.
- [RFC4520] Zeilenga, K., "Internet Assigned Numbers Authority (IANA) Considerations for the Lightweight Directory Access Protocol (LDAP)", BCP 64, RFC 4520, June 2006.

Appendix A. Examples

In this section, we will describe a fictional DUA that 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" object class type [RFC2789]. 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 object class).

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 whitespace-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 its 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))
```

Authors' Addresses

Bob Neal-Joslin (editor)
Hewlett-Packard Company
19420 Homestead RD
M/S 4029
Cupertino, CA 95014
US

Phone: +1 408 447 3044
EMail: bob_joslin@hp.com
URI: <http://www.hp.com>

Luke Howard
PADL Software Pty. Ltd.
PO Box 59
Central Park, Vic 3145
AU

EMail: lukeh@padl.com
URI: <http://www.padl.com>

Morteza Ansari
Infoblox
475 Potrero Avenue
Sunnyvale, CA 94085
US

Phone: +1 408 716 4300
EMail: morteza@infoblox.com

Full Copyright Statement

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

