LDAPEXT Working Group
Internet Draft

B. Greenblatt
Directory Tools and
Application Services, Inc.

<draft-greenblatt-ldapext-style-01.txt>
March 2001
Category: Informational

# LDAP Extension Style Guide

Status of this Memo



## 1. Abstract

Version 3 of the Lightweight Directory Access Protocol (LDAP) as defined in [1] provides a base set of services. Additionally, LDAP provides several mechanisms by which the base set of services may be enhanced to provide additional services. This document describes the different ways that LDAP may be enhanced, and how developers can decide which enhancement mechanism is best suited for their environment. It also discusses the positives and negatives for each LDAP enhancement mechanism

## 2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [2].

## 3. Introduction

There are four mechanisms for enhancing the base set of services offered by LDAP:



- Controls
- Extended Operations
- Schema Enhancements
- New Attribute Type Syntaxes

Each of these enhancement mechanisms will be described separately in the following sections. Each section will include examples that show appropriate usage for that mechanism. Each section also includes examples that show inappropriate usage for that mechanism.

## 4. LDAP Controls

An LDAP Control is a mechanism that allows additional parameters to be added to previously defined LDAP operations. The LDAP operations defined in [1] are:

- Bind
- Unbind
- Search
- Add
- Modify
- ModifyDN
- Delete
- Compare

Each of these operations has a defined set of parameters that are passed in the LDAPMessage construct. A control is the preferred means of enhancing an existing operation. The control mechanism SHOULD be used when it does not fundamentally change the meaning and operating characteristics of an existing operation. LDAP controls have a criticality that is defined. The criticality field is only meaningful when the control is passed from the LDAP client to the LDAP server in the operation request. LDAP clients ignore the criticality field in controls that appear in operation results.

The parameters that appear in the control reside within the controlValue field. The controlValue field is encoded as an octetString. Its value may be defined by the use of a BNF grammar or an ASN.1 syntax definition. If BNF is used, the use of BNF MUST be in conformance with the Augmented BNF definitions of [4]. No preference is given towards either definition.

Whenever a new control is defined for a specific operation request, the specification MUST clearly specify which controls (if any) are allowed to be placed in the corresponding operation result.

Furthermore, the specification SHOULD clearly specify interactions with other, previously-defined, extensions (using other controls). The specification MAY NOT restrict further extension of the operation by placement of additional, yet to be defined, controls."

An example of an LDAP extension that is appropriate for implementation as a control is the sorted results control that can be used in the Search operation as defined by [5]. The Search



operation normally returns all entries that match the supplied filter. The results are returned in any order that is appropriate for the LDAP server. The sorted results control only changes the order in which the matched entries are returned to the LDAP client. The control does not substantially change the way in which the LDAP server implements the Search operation. It is left up to the client to decide on the criticality of the control. Unless there is an overwhelming reason why the Search should not be performed if the sorted results control then the criticality should be FALSE. As a general rule, unless the results of the operation would be useless (or potentially harmful) control criticality SHOULD be given the value FALSE.

The Search request includes a parameter that allows the client to request whether or not aliases are dereferenced. RFC 2256 defines an object class called organizationalRole. This object class is similar to the alias object class in that it includes a roleOccupant attribute that holds one or more distinguished names of other entries in the directory. The search request does not include a parameter to automatically dereference roleOccupants. Thus, it is possible to define a control to request whether or not to dereference roleOccupants. If this request is made, and is supported by the LDAP server, then the LDAP server handles the organizationalRole entries in a manner similar to the way in which it handles alias entries in the presence of the derefAliases parameter. The controlValue can be defined using this ASN.1:

```
derefOrganizationRoles    ENUMERATED {
     neverDerefRoles       (0),
     derefInSearching        (1),
     derefFindingBaseObj     (2),
     derefAlways             (3)}
```

Similarly, it can be defined using this BNF:

```
derefOrganizationalRoles =
     neverDerefRoles | derefInSearching | derefFindingBaseObj |
     derefAlways
neverDerefRoles = ô0ö
```

derefInSearching = ô1ö
derefFindingBaseObj = ô2ö
derefAlways = ô3ö

The handling of the control value is similar to the way in which the derefAliases parameter value is handled. The major difference is that the roleOccupant attribute may be multi-valued, and the Search operation may fan out in multiple directions, which would not be the case with the single-valued aliasedObjectName attribute. The question arises as to whether the dereference organizational role control fundamentally changes the behavior of the Search operation. There is some change in the behavior due to the multiway fan out of the Search operation. So, is this change "fundamental"? In this situation, the answer is no. The behavior is so similar to the



behavior of the derefAliases parameter already in the Search request that the change is not seen to be fundamental.

Consider the LDAP Extension to copy an entry or subtree from one part of the Directory Information Tree (DIT) to another. This extension can be defined as a control in the ModifyDN operation. This operation already moves an entry or subtree from one part of the tree to another. A control can be defined to indicate that instead of moving the entries from one part of the DIT to another, the entries in the named subtree are copied to the new part of the DIT. This control could be defined using this ASN.1:

```
copySubtreeControl ::= SEQUENCE {
     target           LDAPDN,
     filter           Filter OPTIONAL}
```

Similarly, it can be defined using this BNF:

copySubtreeRequest = source SEP target SEP filter
target = LDAPDN ; defined according to [6]
filter = UTF-8String ; defined according to [9]
SEP = ô;ö

If the filter is present in the request, only those objects in the source subtree that match the filter are copied to the target subtree. Even though the copy subtree extension can be defined using this control, it should not be defined this way. This is due to the fact that it fundamentally changes the behavior of the modifyDN operation. As it is currently defined, the modifyDN operation is logically just a change in name that affects the entry named in the ôentryö parameter of the modifyDN operation. The addition of the copy subtree control would fundamentally change this

behavior. Thus, the copy subtree extension should not be implemented as a control, and instead by implemented as an extended operation.

The definition of a control SHOULD be defined in such a manner that it is extensible. For extensibility, extra binary fields SHOULD be built into the definition. In ASN.1, use of a SEQUENCE is helpful. In order to allow for extensibility, the copySubtreeControl can be defined as:

```
copySubtreeControl ::= SEQUENCE {
     target           LDAPDN,
     filter           Filter OPTIONAL,
     extensions   [0] OCTET STRING OPTIONAL}
```

Similarly, using ABNF the request can be defined as:

```
copySubtreeRequest = source SEP target SEP filter SEP extensions
target = LDAPDN ; defined according to [6]
filter = UTF-8String ; defined according to [9]
extensions = binary ; arbitrary binary data
```



Note that servers SHOULD not send back controls in an operation response that have not been requested by the client. In the event that a client does receive an unsolicited control in a response, the client MAY ignore the control.

### 4.1 Controls on the Bind

Special care should be taken when enhancing the Bind operation with controls. Note that controls used in the Bind operation are not protected by the privacy and integrity negotiated by the bind operation itself. This must be taken into consideration. If controls are passed on the Bind that need privacy and/or integrity protection, a TLS session SHOULD be negotiated prior to the Bind.

## 5. LDAP Extended Operations

An LDAP Extended Operation is a mechanism that allows for new LDAP operations to be defined to enhance the base set listed above. The extended operation describes the parameters that are passed in the LDAPMessage construct. The extended operation MUST define both the ExtendedRequest message that is passed from the LDAP client to the LDAP server, as well as the ExtendedResult message that is passed from the LDAP server back to the LDAP client. The extended operation mechanism SHOULD be used when its operating characteristics are fundamentally different from the base set of LDAP operations.

The parameters that appear in the extension reside within the requestValue field. The requestValue field is encoded as an octetString. Its value may be defined by the use of a BNF grammar or an ASN.1 syntax definition. If BNF is used, the use of BNF MUST be in conformance with the Augmented BNF definitions of [4]. No preference is given towards either definition.

Consider the copy subtree extension mentioned above. Since it fundamentally changes the behavior of the base LDAP operations, it will be defined using an extended operation. The requestValue has this ASN.1:

```
copySubtreeRequest ::= SEQUENCE {
     source           LDAPDN,
     target           LDAPDN,
     filter           Filter OPTIONAL}
```

Similarly, it can be defined using this BNF:

```
copySubtreeRequest = source SEP target SEP filter
source = LDAPDN ; defined according to [6]
target = LDAPDN ; defined according to [6]
filter = UTF-8String ; defined according to [9]
```

Notice that the source parameter is added here as opposed to the previously defined control. This is due to the fact that the



previously defined control made use of the ôentryö parameter of the modifyDN operation.

The definition of an extended operation should be defined in such a manner that it is extensible. For extensibility, extra binary fields SHOULD be built into the definition. In ASN.1, use of a SEQUENCE is helpful. In order to allow for extensibility, the copySubtreeRequest can be defined as:

```
copySubtreeRequest ::= SEQUENCE {
     source           LDAPDN,
     target           LDAPDN,
     filter           Filter OPTIONAL,
     extensions   [0] OCTET STRING OPTIONAL}
```

Similarly, using ABNF the request can be defined as:

```
copySubtreeRequest = source SEP target SEP filter SEP extensions
source = LDAPDN ; defined according to [6]
target = LDAPDN ; defined according to [6]
```

```
filter = UTF-8String ; defined according to [9]
extensions = binary ; arbitrary binary data
```

Note that servers MUST not send back extended responses that have not been requested by the client. In the event that a client does receive an unsolicited extended response in a response, the client MAY ignore the extended response.

## 6. LDAP Schema Extensions

The base set of LDAP Object Classes and Attribute Types are defined in [2] and [3]. Schema is the collection of attribute type definitions, object class definitions and other information that a server uses to determine how to match a filter or attribute value assertion (in a compare operation) against the attributes of an entry, and whether to permit add and modify operations. Schema extensions are the preferred mechanism of enhancing LDAP. This is due to the fact that all LDAP servers allow their base schemas to be enhanced. Furthermore, it is a requirement that the LDAP server MUST publish the schema supported by an LDAP server.

New attribute types MUST not be added to existing object classes. New object classes that are defined SHOULD use existing attribute types when the data elements are substantially similar to existing data elements that have previously been defined. The use of schema extensions allows normal LDAP operations to be used to supply enhancements to the set of base LDAP services. For example, the sorted results control was previously mentioned. This control is only useful in dealing with whole attributes that appear within entries.

Consider a search that wants to retrieve the list of users by location code (a subfield of the phone number), but sorted by surname within location code. The telephoneNumber attribute type is



defined as a character string that is assumed to contain the location code. Unfortunately, the location code is not broken out from the telephone number, so it is not generally possible to algorithmically determine the location code from examining the telephone number attribute. Furthermore, the telephoneNumber is a multi-valued attribute. Each attribute value might contain a logically different location code. In order to adequately support this feature, a new attribute type can be defined to hold the primary location code of the entry. This can be defined as follows:

```
(tempOID NAME 'locationCodeInformation' SUP top AUXILIARY
     MUST primaryLocationCode )
```

(tempOID NAME 'primaryLocationCode' SUP ænameÆ SINGLE-VALUE )

Note that real object identifiers are not used in the above definitions, since this document is not actually defining the locationCodeInformation object class. The desired sort can now be achieved by using the primaryLocationCode attribute type and the surname attribute type within the sort results control. This sort only works if the locationCodeInformation is populated within the DIT. The LDAP server does not automatically populate the primaryLocationCode using other attributes, so it is incumbent upon the LDAP client to populate the primaryLocationCode attribute if the sort is to work as desired.

## 7. New Attribute Type Syntaxes

The base LDAP Syntaxes are defined in [2]. It is occasionally the case that there is no defined syntax that exactly matches a previously defined syntax. When this circumstance arises, there are two alternatives:

- Define a new attribute syntax
- Use a binary syntax, and define a BNF grammar for the attributes that fits inside the binary syntax.

Either of these alternatives defines new attribute syntaxes. The use of BNF is preferred in environments where the LDAP Server is not specifically required to understand the syntax. Furthermore, there is no requirement of compliant LDAP servers to be able to support attribute type syntaxes that are defined outside of [2]. Thus, the use of BNF on top of existing attribute syntaxes is preferred as it is more likely to be interoperable among LDAP servers supplied form multiple sources. The use of BNF MUST be in conformance with the Augmented BNF definitions of [4]. When a binary syntax is chosen, the Octet String syntax defined in [2] which uses 1.3.6.1.4.1.1466.115.121.1.40 as the object identifier SHOULD be used as the wrapper attribute syntax. When the data to be stored is character data, the Directory String syntax defined in [2], which uses 1.3.6.1.4.1.1466.115.121.1.15 as the object identifier SHOULD be used instead.



Since not all LDAP servers support (or easily support) the addition of new attribute type syntaxes, the use of the attribute type syntaxes is not always available. The use of an Octet String or a Directory String in combination with BNF is normally a better alternative, and SHOULD be used. In defining the BNF, strong consideration should be paid to matching rules. In string data, the

BNF SHOULD be defined so that the substring matching rule is still effective. For example, [2] defines the postal address syntax as a Directory String syntax that uses the following BNF:

postal-address = dstring *( "$" dstring )

An example character string using this BNF is:

1234 Main St.$Anytown, CA 12345$USA

This definition allows substring matching to still be effective, especially through its use of separation characters. For example, the ô$Anytownö string could be used as the ôANYö sub-filter to find all entries with a postal address in Anytown. Use of this sub-filter would not match those entries that have a postal address on ôAnytown Blvdö that are not actually in he city Anytown.

Use of ASN.1 for new attribute type syntaxes SHOULD only be used in the case of very complex data types, and only then after serious consideration of an ABNF representation. Whenever ASN.1 is used for specifying a new attribute type syntax, the ASN.1 encoding mechanism MUST also be specified (DER encoding is STRONGLY preferred).

## 8. The Grey Area

In some situations it is not clear whether to use a control or an extended operation. Consider an LDAP extension that would delete an entire subtree instead of deleting a single entry as the current Delete operation does. This could be implemented as either a control used with the existing Delete operation or a new extended operation. In fact Internet drafts have been proposed using both methodologies [7], [8]. Persuasive arguments can be made about implementing this LDAP extension either as an extended operation or a control. An operation on a subtree is different than an operation than an operation on an individual entry. But, are the operations different enough to implement the subtree delete operation as an extended operation.

A good guideline to use for deciding if the use of a control is appropriate is to examine what would happen if an LDAP server that did not support it received the control. If the criticality field is set to TRUE, then the LDAP server would return the error code unsupportedCriticalExtension. If the criticality field is set to FALSE, then the LDAP server ignores the control and operates on the remainder of the LDAP operation request. If the LDAP server attempts to implement the operation with the non-critical control and would always return an error result code, then the LDAP

extension SHOULD NOT be implemented as a control, but instead SHOULD be implemented as an extended operation.

If there are many scenarios in which an LDAP server ignoring a non-critical control would still be able to successfully implement the operation, then the LDAP extension SHOULD be implemented as a control. If there are only a few scenarios in which an LDAP server ignoring a non-critical control would still be able to successfully implement the operation, then consensus should be sought from the LDAP community. The smaller the number of valid scenarios in which an LDAP server ignoring a non-critical control would still be able to successfully implement the operation, then greater consideration should be given to the use of an extended operation. Similarly, the greater the number of valid scenarios in which an LDAP server ignoring a non-critical control would still be able to successfully implement the operation, then greater consideration should be given to the use of a control. In the case of the subtree delete extension, the only scenario in which an LDAP server ignoring the control would still be able to successfully implement the delete operation is when the entry named in the DelRequest is a leaf entry. Using the guideline mentioned above, consideration should be given to the use of an extended operation for the implementation of the delete subtree extension.

## 9. Security Considerations

Implementors and administrators should be aware that à

## 10. References


1 Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

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

[1] Wahl, M., Kille, S. and Howes, T., "Lightweight Directory Access Protocol (v3)", Internet Standard, December, 1997. RFC2251.

[2] Wahl, M., Coulbeck, A., Howes, T. and Kille, S., "Lightweight Directory Access Protocol (v3), Attribute Syntax Definitions", Internet Standard, December, 1997. RFC2252.

[3] Wahl, M., "A Summary of the X.500(96) User Schema for use with LDAPv3", Internet Standard, December, 1997. RFC2256.

[4] Crocker, D., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[5] Herron, A., et. al, "LDAP Control Extension for Server Side

Sorting of Search Resultsö, RFC NNNN, July 2000.



[6] Wahl, M., Kille, S. and Howes, T., "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", Internet Standard, December, 1997. RFC2253.

[7] Greenblatt, B, ôSimple Operations on Subtrees (for LDAP)ö, Internet Draft (Work in Progress), August 2000, draft-greenblatt-ldapext-sos-01.

[8] Armijo, M, ôLDAP Tree Delete Controlö, Internet Draft (Work in Progress), expired, ftp://ftp.isi.edu/internet-drafts/draft-armijo-ldap-treedelete-03.txt.

[9] Howes, T., "The String Representation of LDAP Search Filters", Internet Standard, December, 1997. RFC2254.

## 10. Acknowledgments

Thanks to Kurt Zeilenga for an informal review prior to submission.

## 11. Author's Addresses

Bruce Greenblatt
Directory Tools and Application Services, Inc.
6841 Heaton Moor Drive
Phone: +1-408-390-4776
Email: bgreenblatt@directory-applications.com

Full Copyright Statement