December 1998
Volume 1, Issue 5

Internet Exchange News

Inside this Issue

1 Sending and receiving email over the Internet without the hassles

2 The Lightweight Directory Access Protocol (LDAP): An overview

10 Questions and Answers

This Month's Tips:

- The Fully Qualified Domain Name
- Upgrading Internet Exchange 3.0 to 3.11

International Messaging Associates (IMA) Ltd.
Hong Kong Computer Center
54-62 Lockhart Road
Wan Chai, Hong Kong
Tel: +852 2520-0300
Fax: +852 2648-5913

IMA Philippines Inc.
The Peak Tower
15/F 107 Alfaro Street
Salcedo Village
Makati City, Philippines
Tel: +63 (2) 811-3999
Fax: +63 (2) 811-3939

US Support: +1 (408) 481- 9985
US Sales: +1 (408) 481- 9985
US Fax: +1 (888) 562 - 3561

Email : info@ima.com
Website: www.ima.com

CASE STUDY

Sending and receiving email over the Internet without the hassles; *IMA's Internet Exchange provides the State Government of Indiana with a robust messaging tool for the Information Age*

With approximately 40,000 employees in its roster, the Indiana State Government is indeed a very large and complex organization. The seat of government is located in the city of Indianapolis, and from there it governs the affairs of the entire state, excluding those handled by the federal and local governments, through more than 800 offices and 70 state agencies, such as the Indiana Department of Administration, the Indiana Arts Commission, the Family and Social Services Administration, the State Student Assistance Commission of Indiana, the Indiana Department of Tourism, the department of Workforce, the Bureau of Motor Vehicles, and the Attorney General's Office. Its responsibilities range from the issuing of welfare checks to building highways and maintaining driver registrations and public health records, to the development of various education and tourism programs.

With so many responsibilities in hand, it is not surprising that the Indiana State Government is now making use of advanced information and communications technology that will help maintain good governance throughout the state. One such technology is Internet electronic mail.

"The government of the state of Indiana is committed in employing the best available resources to perform its many functions," says Raymond L. Gilbert, systems administrator. According Gilbert, the Indiana State Government has approximately 15,000 email users with full Internet connectivity. Around 3,500 of them use cc:Mail and are connected to the Internet via IMA's Internet Exchange gateway. Most department and agencies within the state have their own IT staff concentrating on technologies that they might need to perform their duties. They report to a central IT department, the Information Services Division, and to the Data Processing Oversight Commission.

"Before we installed Internet Exchange in 1994, we had the vast majority of our users on a mainframe or midrange system. The growing number of cc:Mail users were connected to the rest of the statewide email system via a product from SoftSwitch that ran on the mainframe and routed messages between various systems," says Gilbert. "Shortly after we set up an Internet connection for our network, it was decided that cc:Mail users must be able to send and receive messages to and from the Internet. Internet Exchange was installed to provide our cc:Mail users with Internet connectivity."

Continued on page 13

The Lightweight Directory Access Protocol (LDAP): An overview

Introduction

As computer networking increasingly became popular in the 1980s, it became apparent that a global electronic directory service (EDS) was needed to optimize the potential of distributed computer systems. To achieve this goal, the International Standards Organization (ISO) and the International Telegraph collaborated with the Telephone Consultative Committee (CCITT) to develop the X.500 directory standard within the Open Systems Interconnection (OSI) model in 1988. From this standard evolved the Lightweight Directory Access Protocol (LDAP), which was developed at the University of Michigan.

The X.500 directory standard was developed to provide the networking community with an online version of the "white pages" or "yellow pages." Such online directories are very useful, particularly to Internet and intranet users. They can be used to search for a person or an organization's e-mail address or other important information (i.e. organization names, department names, telephone numbers, etc.). On the Internet, the Domain Name System serves as the directory for relating a domain name to a particular network (IP) address. For a user to send email via the Internet, he must first know the email address of the recipient. But what if he does not know the email address of the intended recipient? This is just one of the issues that the X.500 directory standard seeks to address by providing a directory that contains a wide range of information, such as peoples' names, company names, telephone numbers, and postal codes.

How the X.500 standard works

The X.500 EDS is based on a client-server architecture (see Figure 1). It consists of three major components: the Directory System Agent (DSA), the Directory User Agent (DUA), and the Directory Information Base (DIB). The DSA resides on the server computer and manages information in the directory, while the DUA is an application software component that allows the client computer to access the directory. The DIB serves as the repository for information.

Aside from these three key components, the X.500 EDS also uses several protocols for managing information. The Directory Access Protocol (DAP) facilitates communication between DUA's and DSA's, while the Directory System Protocol (DSP) supports communication among DSA's for distributed directory operations. The Directory Information Shadowing Protocol (DISP), on the other hand, allows two or more DSA's to share copies of their DIBs for replication purposes. Another protocol, the Directory Operations Protocol (DOP), is also used for replication purposes. However, the DOP is seldom used in X.500 EDS's. The directory is distributed among several DSA's, and if one of the DSA's fails to answer a client's request, the client is

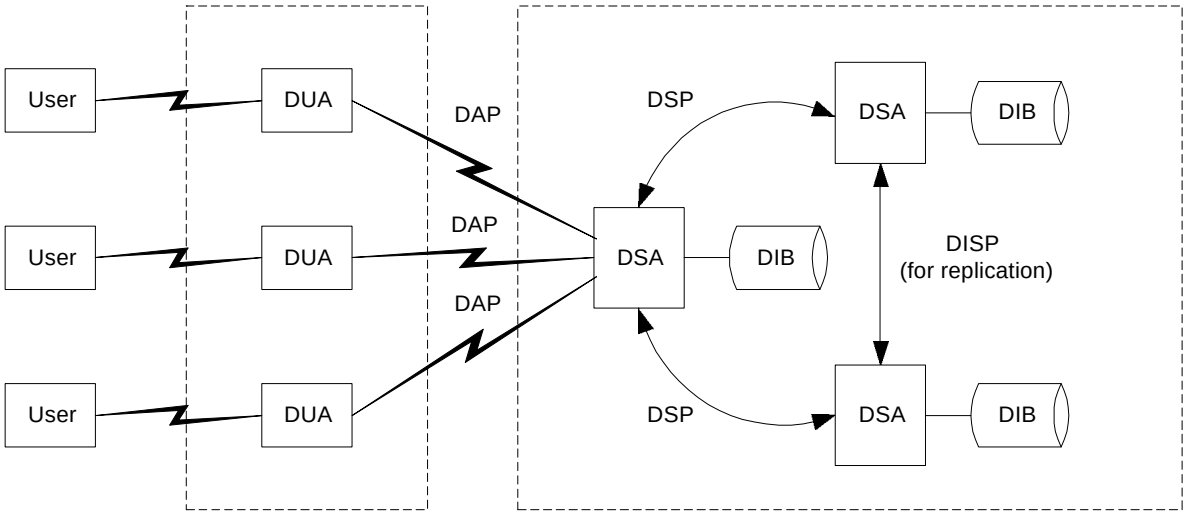

Figure 1. The X.500 Electronic Directory Service Architecture

referred to another DSA, a process called chaining.

In an X.500 EDS, information is stored using an organizational tree-like structure. This is called the Directory Information Tree (DIT) (see Figure 2).

Object classes define the structure of an entry in the directory. There are 17 basic object classes defined in the X.500 standard. These are:

- Alias
- Country
- Locality
- Organization
- Organizational Unit
- Person
- Organizational Person
- Organizational Role
- Residential Person
- Group of Names
- Group of Unique Names
- Application Process
- Application Entity
- DSA
- Device
- Strong Authentication User
- Certification Authority

Objects belonging to these classes are identified by their attributes, which determine what kind of information can be stored in the directory. These attributes include:

- Common Name
- Surname
- Given Name
- Organizational Unit Name
- Organizational Name
- Aliased Object Name
- Knowledge Information
- Country Name
- Locality Name
- State or Province Name
- Telephone Number
- Postal Code
- Post Office Box
- Registered Address

An attribute is a type with one or more associated values. The attribute type is identified by a short descriptive name and an OID (object identifier). The attribute type governs whether there can be more than one value of an attribute of that particular type of entry, the syntax to which the values must conform, the kinds of matching which can be performed on the values of that attribute, and other function.

Schema is the collection of attribute type definitions, object class definitions and other information which 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 as requested by the client.

Directory entries are represented as nodes in the DIT. the topmost node is known as the “root” (see Figure 2). Each entry can be uniquely identified by its *Distinguished Name* (DN). For example, in Figure 2 the person Tim Kehres at the Engineering Department of IMA Inc. has the DN “CN=Kehres, OU=Engr, O=IMA, C=US”. Thus, the name Kehres can be entered as many times in the DIT provided it is located under different entries so it will have different DN's. Each component of the DN is called the *Relative Distinguished Name* (RDN).

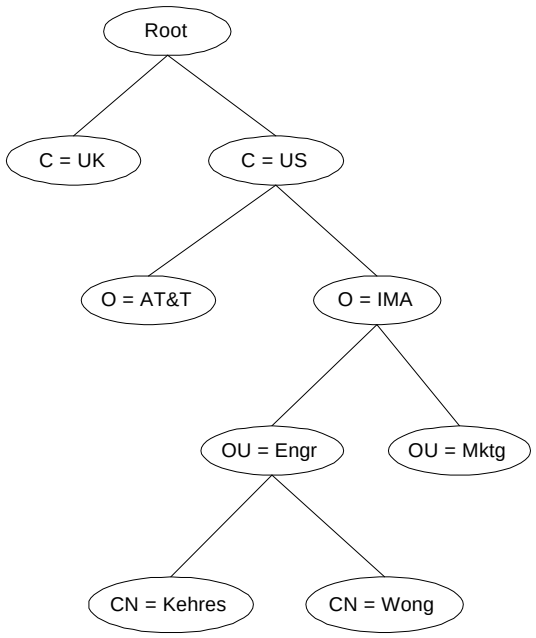

Figure 2. X.500 EDS Hierarchy

To ensure a logical hierarchy of the different types of entries, the DIT structure is enforced by Name Bindings. Object class definitions and attribute specifications, on the other hand, define the internal structure of the entries. Thus, a Country entry is prevented from being included in the Person Name entry and vice-versa.

The LDAP Advantage

The DAP allows communication between the DUA and the DSA. It provides a wide range of directory services via such commands as *Read, Compare, Aban-*

don, Add Entry, Remove Entry, Modify Entry, Modify RDN, Search, and *List.*

However, DAP is a complete OSI application, meaning it requires all layers of the 7-layer OSI protocol stack. Plus it has too much code and requires excessive computing horsepower to run. These factors place an unreasonable burden on client machines that are usually not designed to support OSI protocol stack, thereby, posing a limitation to intranet and Internet users.

To provide such machines with directory-access capability, the University of Michigan developed the Lightweight Directory Access Protocol (LDAP), a simplified version of the DAP. Since then, the protocol has gone through several revisions. RFC 1777 defines LDAP version 2 as a protocol for providing read/write access to X.500 directories using less processing requirements than the DAP. LDAP2 supports simple authentication using a cleartext password as well as the Kerberos version of authentication. It can also run over a Secure Socket Layer/Transport Layer Security (SSL/TLS) transport layer.

The latest of LDAP is version 3 (LDAP3). As defined in RFC 2251, LDAP3 is a protocol designed to provide access to open X.500 directory service and proprietary directories that support the X.500 standard without incurring the resource requirements of the DAP. It is specifically targeted at management and browser applications that provide read/write interactive access to directories. Like LDAP2, LDAP3 is designed to complement the DAP when used with a directory that supports the X.500 protocols. LDAP3 supports all the security features found in LDAP2. In addition, it supports the SASL, which allows an extensible authentication and security framework. LDAP3 is also designed to return referrals to other servers to clients.

LDAP does not require the upper layers of the OSI protocol stack and runs directly on TCP/IP or other reliable transport protocols. Moreover, it uses a smaller amount of code than the DAP and requires minimal processing overhead. LDAP uses simple character strings to encode DN's and data elements (see RFC 2252, RFC 2253, and RFC 2254), as compared to the X.500, which uses highly-structured encoding even for simple data elements. This makes it easier to decode large DN's. The protocol also eliminates the need for the read and list operations, emulating them by means of the search operation.

To gain access to X.500 directories, an LDAP server must be able to support both TCP/IP and OSI protocols. This type of server answers the needs of the client by becoming a client to the X.500 server (see Figure 3). LDAP servers designed to access proprietary directories are not required to support the OSI protocols. Such servers are known as stand-alone LDAP servers since they do not rely on X.500 servers (see Figure 4).

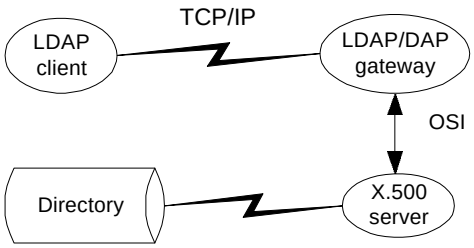

Figure 3. LDAP server accessing an X.500 directory

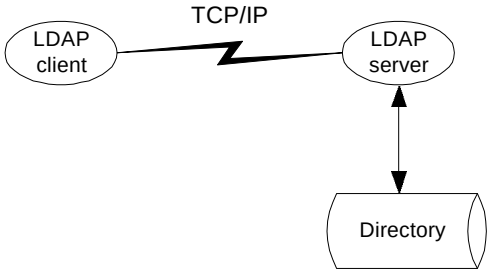

Figure 4. Stand-alone LDAP server

The LDAP session

As stated in RFC 2251, the LDAP client may initiate the session by binding to the LDAP server. Or it may request the binding process and request for any operations. It provides the IP address or host name as well as the TCP/IP port number where the LDAP3 server is listening. The client also indicates the LDAP version it supports during the bind request. If no LDAP version is indicated by the client, the server will assume that the client supports LDAP3.

In LDAP2, the client can give a user name and a corresponding password to identify itself to the server. However, this is strongly discouraged in case the underlying transport cannot guarantee confidentiality. In LDAP3, the SASL authentication scheme is supported to provide security services to clients. For some SASL authentication mechanisms, the client may have to invoke bind requests several times. Once the session has been established, the client can manipulate or query information contained in the directory using the LDAP operations. After the client finished making requests, it issues an unbind request to the server to close the session.

During an LDAP session, LDAPMessage Protocol

Data Units (PDU's) are mapped directly onto the TCP bytestream. LDAP servers provide a protocol listener on port 389 (although a different port number may be used). LDAP clients, on the other hand, connect to the server on any valid TCP port.

LDAP clients that request for referrals are not allowed to loop between servers. They must not repeatedly contact the same server for the same request with the same target entry name, scope, and filter.

The LDAP3 elements

LDAP3 consists of a number of protocol elements used in various protocol operations. A brief description of each of these elements follows.

1. **Message Envelope** - all protocol operations are encapsulated in a common envelope known as the LDAPMessage. This envelope contains common fields required in all protocol exchanges. This envelope is defined as follows:

```
LDAPMessage  : : = SEQUENCE  {
 MessageID            MessageID
 protocolOP             CHOICE  {

 bindRequest            BindRequest,
 bindResponse           BindResponse,
 unbindRequest          Unbindequest,
 searchRequest          SearchRequest,
 searchResponse         SearchResponse,
 modifyRequest          ModifyRequest,
 addRequest             AddRequest,
 delRequest             DelRequest
 delResponse            DelResponse,
 modifyDNRequest        ModifyDNRequest,
 modifyDNResponse       ModifyDNResponse,
 compareRequest         CompareRequest,
 abandonRequest         AbandonRequest,
 extendedReq            ExtendedRequest,
 extendedResp           ExtendedResponse},

controls     [0]       Controls  OPTIONAL  }

MessageID  : : =  INTEGER  (0 .. maxInt)
maxInt Integer : : 2147483647 -- (2^^31 - 1) --
```

If the server receives a PDU in which the LDAPMessage SEQUENCE tag cannot be recognized, the tag of the protocolOp is not recognized as a request, the message ID cannot be parsed, or the encoding structures or lengths of data fields are incorrect, the server then returns a notice of disconnection and terminates the connection with the client.

a. Message ID – Every LDAPMessage envelope that encapsulates responses contains the messageID value of the corresponding LDAPMessage. This parameter must not have a value similar to the values of any other outstanding requests in the LDAP session of which the said message is a part. A client cannot send a second request whose messageID is the same as a previous request on the same connection, unless the client has already received a final response from the server regarding the earlier request.

2. **String Types** – this indicates that the ISO 10646 character set, a superset of Unicode, is used and encoded following the UTF-8 algorithm.

3. **DN and RDN** – A LDAPDN and a RelativeLDAPDN represent a Distinguished Name (DN) and a Relative Distinguished Name (RDN), respectively. These parameters are obtained after encoding the DN and RDN using the Abstract Syntax Notation 1 (ANS.1) Basic Encoding Rules (BER) such that:

<distinguished-name> : : = <name>
<relative-disntinguished-name> : : = <name-component>

Note: Definitions of the <name> and <name-component> parameters can be found in RFC 2253.

Only Attribute Types are allowed in an RDN component. It must also be noted that the options of Attribute Types must not be used in specifying DN's.

4. **Attribute Type** – the value of this parameter is the textual string associated with that AttributeType in its specification. For example, the Attribute type "organizationName" with object identifier 2.5.4.10 is represented as an AttributeType in this protocol by the string "organizationName". If there is no textual name for an attribute type, then the object identifier is used by the protocol.

A specification may assign more than one textual name to an attribute type. These names must begin with a letter and must contain ASCII letters, digit characters and hypens only. They are

also case sensitive.

5. **Attribute Description** – this is a superset of the definition of the AttributeType. It allows additional options to be specified. The following is an example of a valid AttributeDescription:

   cn
   userCertificate;binary

   The value of the AttributeDescription is based on the Backus-Naur Form (BNF) notation. Any option can be associated with any AttributeType. However, not all combinations may be supported by the server. An AttributeDescription with one or more options is treated as a subtype of the attribute type without any options. A server will treat an AttributeDescription with any options it does not implement as an unrecognized attribute type.

   If the "binary" option is present in an AttributeDescription, it overrides any string-based encoding representation defined for that attribute by RFC 2252. Instead, the attribute will be transferred as a binary value encoded using the Basic Encoding Rules described in ITU-T Rec. X.690. The syntax of the binary value is an ASN.1 data type definition which is referenced by the "SYNTAX" part of the attribute type definition.

6. **Attribute Value –** this parameter takes on its value either a string encoding of a AttributeValue data type, or an octet string containing an encoded binary value, depending on whether the "binary" option is present in the companion AttributeDescription to this AttributeValue. The definition of string encodings for different syntaxes and types may be found in other documents, and in particular in RFC 2252.

   Attributes may be defined which have arbitrary and non-printable syntax. Implementations must neither simply display nor attempt to decode as ASN.1 a value if its syntax is not known. The implementation may attempt to discover the subschema of the source entry, and retrieve the values of attributeTypes from it. LDAP clients are not permitted to send attribute values in a request which are not valid according to the syntax defined for the attributes.

   Note: There is no maximum limit for the size of this value's encoding.

7. **Attribute Value Assertion –** this parameter contains an attribute description and a matching rule assertion value suitable for that type. It is defined as follows:

   ```
   AttributeValueAssertion ::= SEQUENCE {
           attributeDesc AttributeDescription,
           assertionValue AssertionValue }

   AssertionValue ::= OCTET STRING
   ```

   If the "binary" option is present in attributeDesc, this informs the server that the assertionValue is a binary encoding of the assertion value. For all the string-valued user attributes described in RFC 2252, the assertion value syntax is the same as the value syntax. Clients may use attribute values as assertion values in compare requests and search filters. Note, however, that the assertion syntax may be different from the value syntax for other attributes or for non-equality matching rules. These may have an assertion syntax which contains only part of the value.

8. **Attribute** – this parameter consists of a type and one or more values of that particular type. It is defined as follows:

   ```
   Attribute : : =  SEQUENCE {
           type        AttributeDescription,
           vals        SET OF AttributeValue }
   ```

   Please take note that each attribute value is distinct in the set and that the order of attribute values within the vals set is undefined and implementation dependent.

9. **Matching Rule Identifier** – the matching rule defines the syntax of the AssertionValue and the process that must be performed in the server. It provides a means for expressing how a server should compare an assertion value received in a search filter with an abstract data value.

   MatchingRuleId : : = LDAPString

   Servers that support matching rules for use in the extensibleMatch search filter are required to list the matching rules they implement in subschema entries, using the matchingRules attributes. They must also list the attribute types with which each matching rule can be used, using the matchingRuleUse attribute.

10. **Result Message** – the LDAPResult is the construct used by the server to inform the client whether a command or request return has been completed successfully or not. Some of these responses are:

    success
    operationsError
    protocolError
    timeLimitExceeded
    sizeLimitExceeded

noSuchAttribute
authMethodNotSupported
objectClassViolation
compareFalse
compareTrue

All the result codes, with the exception of *success*, *compareFalse*, and *compareTrue*, mean that the operation cannot be competed in its entirety.

11. **Referral –** the referral error indicates that the contacted server does not hold the target entry of the request issued by the client. The referral field is present in an LDAPResult if the LDAPResult.resultCode field value is referral, and absent with all other result codes. It contains a reference to another server (or set of servers) which may be accessed via LDAP or other protocols. Referrals can be returned in response to any operation request (except unbind and abandon which do not have responses). At least one URL must be present in the referral.

    The referral is not returned for a singleLevel or wholeSubtree search in which the search scope spans multiple naming contexts, and several different servers would need to be contacted to complete the operation. If the client wishes to continue the operation, it must follow the referral by contacting any one of servers. All the URLs should be equally capable of being used for the operation to continue. URLs for servers implementing the LDAP protocol are written according to RFC 2255.

12. **Controls** – a control is a way to specify extension information. Controls which are sent as part of a request apply only to that request.

    ```
    Controls ::= SEQUENCE OF Control
    Control ::= SEQUENCE {
    controlType         LDAPOID,
    criticality         BOOLEAN DEFAULT FALSE,
    controlValue        OCTET STRING OPTIONAL }
    ```

    The controlType field must be a UTF-8 encoded dotted-decimal representation of an object identifier that uniquely identifies the control. This prevents conflicts between control names. The criticality field is either TRUE or FALSE. If the server recognizes the control type and it is appropriate for the operation, the server will make use of the control when performing the operation. If the server does not recognize the control type and the criticality field is TRUE, the server must not perform the operation. Instead it should return the resultCode *unsupportedCriticalExtension*. The same rule applies if the control is not appropriate for the operation and criticality field is TRUE. If the control is unrecognized or inappropriate but the criticality field is FALSE, the LDAP server must ignore the control.
    The controlValue contains any information associated with the control, and its format is defined for the control. The server MUST be prepared to handle arbitrary contents of the controlValue octet string, including zero bytes. It is absent only if there is no value information which is associated with a control of its type. LDAP Servers list the controls which they recognize in the supportedControl attribute in the root DSE.

LDAP Operations

There are several operations involved in an LDAP session. These operations allow the server and the client to communicate with each other so that the tasks requested by the client can be performed by the server.

1. **Bind Operation –** this function allow authentication information to be exchanged between the client and server. The Bind Request is defined as follows:

    ```
    BindRequest::= [APPLICATION 0] SEQUENCE {
      version                 INTEGER (1 .. 127),
       name                     LDAPDN,
       authentication          AuthenticationChoice }

    AuthenticationChoice ::= CHOICE {
       simple [0]               OCTET STRING,
                                -- 1 and 2 reserved
    sasl [3]                    SaslCredentials }

    SaslCredentials ::= SEQUENCE {
       mechanism             LDAPString,
       credentials           OCTET STRING
                                   OPTIONAL }
    ```

    The parameters of the Bind Request are:

    version – this number indicates the version number of the LDAP to be used in the session. If the client requests protocol version 2, a server that supports the version 2 protocol will not return any v3-specific protocol fields.

    name – this refers to the name of the directory object that the client wishes to bind as. This field may take on a null value (a zero length string) for the purposes of anonymous binds, when authentication has been performed at a lower layer, or when using SASL credentials with a mechanism that includes the LDAPDN in the credentials.

authentication – information used to authenticate the name, if any, provided in the Bind Request. After receiving a Bind Request, the LDAP server will authenticate the requesting client, if necessary. It will then return a Bind Response to the client indicating the status of the authentication. If the bind was successful, the resultCode will be *success*, otherwise it will either be:

operationsError – the server encountered an internal error.

protocolError – unrecognized version number or structure.

authMethodNotSupported – unrecognized SASL mechanism.

strongAuthRequired – the server insists that authentication be performed using a SASL mechanism.

referral – the contacted server cannot accept the bind and the client should try another server.

saslBindInProgress – the server insists that the client sends a new bind request with the same authentication mechanism in order for the authentication process to continue.

inappropriateAuthentication – the server issues this response when a client attempts to bind anonymously or without supplying credentials.

invalidCredentials – either the wrong password was given by the client or the SASL credentials cannot be processed.

unavailable – the server is shutting down.

If the server does not support the client's requested protocol version, it will send *protocolError*. After receiving this response, the client must close the connection as the server will be unwilling to accept further operations.

For some SASL authentication mechanisms, it may be necessary for the client to invoke the BindRequest multiple times. If at any stage the client wishes to abort the bind process, it can unbind and then drop the underlying connection. Clients are not allowed to invoke operations between two bind requests made as part of a multi-stage bind. A client may abort a SASL bind negotiation by sending a BindRequest with a different value in the mechanism field of SaslCredentials, or an AuthenticationChoice other than sasl. If the client sends a BindRequest with the sasl mechanism field as an empty string, the server will return a BindResponse with *authMethodNotSupported* as the resultCode. This will allow clients to abort a negotiation if it wishes to try again with the same SASL mechanism.

If the client did not bind before sending a request and receives an *operationsError*, it may then send a Bind Request. If this also fails or the client chooses not to bind on the existing connection, it will close the connection, reopen it and begin again by first sending a PDU with a Bind Request. This will aid in interoperating with servers implementing other versions of LDAP.

Clients may send multiple bind requests on a connection to change their credentials. A subsequent bind process has the effect of abandoning all operations outstanding on the connection (this is to simplify server implementation.) Authentication from earlier binds are subsequently ignored, and so if the bind fails, the connection will be treated as anonymous. If a SASL transfer encryption or integrity mechanism has been negotiated and that mechanism does not support the changing of credentials from one identity to another, then the client must instead establish a new connection.

2. **Unbind Operation** – this function terminates an LDAP session. It is defined as follows:

   UnbindRequest ::= [APPLICATION 2] NULL

   There is no response defined for the Unbind Operation. Upon transmission of an UnbindRequest, the client may assume that the session has been terminated. Likewise, the server may assume that the requesting client has terminated the session and that all outstanding requests may be discarded after an UnbindRequest is received.

3. **Unsolicited Notification –** An unsolicited notification is an LDAPMessage sent from the server to the client which is not in response to any LDAPMessage received by the server. It is used to signal an extraordinary condition in the server or in the connection between the client and the server.

   The notification is of an advisory nature, and the server will not expect any response to be returned from the client. The unsolicited notification is structured as an LDAPMessage in which the messageID is 0 and protocolOp is of the extendedResp form. The responseName field of the ExtendedResponse is present. The LDAPOID value must be unique for this notification, and not be used in any other situation.

4. **Notice of Disconnection –** this notification may

be used by the server to advise the client that the server is about to close the connection due to an error condition. Please take note that this notification is not a response to an unbind requested by the client. It is intended to assist clients in distinguishing between an error condition and a transient network failure. As with a connection closed due to network failure, the client must not assume that any outstanding requests which modified the directory have succeeded or failed.

The following resultCode values are to be used in this notification:

protocolError – the server has received data from the client in which the LDAPMessage structure could not be parsed.

strongAuthRequired – the server has detected that an established underlying security association protecting communication between the client and server has unexpectedly failed or has been compromised.

unavailable – this means that the server will stop accepting new connections and operations on all existing connections, and be unavailable for an extended period of time. The client may make use of an alternative server.

After sending this notice, the server must close the connection. After receiving this notice, the client must not transmit any further on the connection, and may abruptly close the connection.

5. **Search Operation** – The search operation enables the client to request that a search be performed on its behalf by the server. This function can be used to read attributes from a single entry, from entries immediately below a particular entry, or a whole subtree of entries.

The Search Request is defined as follows:

```
SearchRequest::=[APPLICATION3]SEQUENCE{
   baseObject      LDAPDN,
   scope             ENUMERATED{
   baseObject     (0),
   singleLevel     (1),
   wholeSubtree (2) },
   derefAliases   ENUMERATED {
   neverDerefAliases       (0),
   derefInSearching        (1),
   derefFindingBaseObj  (2),
   derefAlways              (3) },
   sizeLimit                INTEGER (0 .. maxInt),
   timeLimit                INTEGER (0 .. maxInt),
   typesOnly                BOOLEAN,
   filter Filter,
   attributes               AttributeDescriptionList }
Filter ::= CHOICE {
   and     [0] SET OF Filter,
   or                  [1] SET OF Filter,
   not                 [2] Filter,
   equalityMatch     [3] AttributeValueAssertion,
   substrings          [4] SubstringFilter,
  greaterOrEqual    [5] AttributeValueAssertion,
   lessOrEqual        [6] AttributeValueAssertion,
   present             [7] AttributeDescription,
   approxMatch       [8] AttributeValueAssertion,
   extensibleMatch  [9] MatchingRuleAssertion }

SubstringFilter ::= SEQUENCE {
   type AttributeDescription,
   substrings          SEQUENCE OF CHOICE {
   initial             [0] LDAPString,
   any                 [1] LDAPString,
   final               [2] LDAPString } }

MatchingRuleAssertion ::= SEQUENCE {
  matchingRule [1] MatchingRuleId OPTIONAL,
   type [2] AttributeDescription OPTIONAL,
   matchValue [3] AssertionValue,
   dnAttributes[4] BOOLEAN DEFAULT FALSE }
```

The parameters of the Search Request are:

baseObject – an LDAPDN that is the base object entry relative to which the search is to be performed.

scope – an indicator of the scope of the search to be performed by the server. The semantics of the possible values of this field are identical to the semantics of the scope field in the X.511 Search Operation.

derefAliases – this parameter indicates as to how alias objects (as defined in X.501) are to be handled in searching.

sizelimit – a sizelimit that restricts the maximum number of entries to be returned as a result of the search. A value of 0 in this field indicates that no client-requested sizelimit restrictions are in effect for the search. Servers may enforce a maximum number of entries to return.

timelimit – a timelimit that restricts the maximum time (in seconds) allowed for a search. A value of 0 in this field indicates that no client-requested timelimit restrictions are in effect for the search.

typesOnly – an indicator as to whether search

results will contain both attribute types and values, or just attribute types. Setting this field to TRUE causes only attribute types (no values) to be returned. Setting this field to FALSE causes both attribute types and values to be returned.

filter – a filter that defines the conditions that must be fulfilled in order for the search to match a given entry.

attributes – a list of the attributes to be returned from each entry which matches the search filter. There are two special values which may be used: an empty list with no attributes, and the attribute description string "*". Both of these signify that all user attributes are to be returned (the "*" allows the client to request all user attributes in addition to specific operational attributes.

Attributes must be named at most once in the list, and are returned at most once in an entry. If there are attribute descriptions in the list which are not recognized, they are ignored by the server.

The results of the search attempted by the LDAP server upon receipt of a Search Request are returned in Search Responses, which are LDAP messages containing either SearchResultEntry, SearchResultReference, ExtendedResponse or SearchResultDone data types.

If the LDAP session is operating over a connection-oriented transport such as TCP, the server will return to the client a sequence of responses in separate LDAP messages. There may be zero or more responses containing SearchResultEntry, one for each entry found during the search. There may also be zero or more responses containing SearchResultReference, one for each area not explored by this server during the search. The SearchResultEntry and SearchResultReference PDU's may come in any order.

Following all the SearchResultReference responses and all SearchResultEntry responses to be returned by the server, the server will return a response containing the SearchResultDone, which contains an indication of success, or detailing any errors that have occurred. Each entry returned in a SearchResultEntry will contain all attributes, complete with associated values if necessary, as specified in the attributes field of the Search Request. Return of attributes is subject to access control and other administrative policy. Some attributes may be returned in binary format (indicated by the AttributeDescription in the response having the binary option present). Some attributes may be constructed by the server and appear in a SearchResultEntry attribute list, although they are not stored attributes of an entry. Clients must not assume that all attributes can be modified, even if permitted by access control. LDAPMessage responses of the ExtendedResponse form are reserved for returning information associated with a control requested by the client. These may be defined in future RFC's.

If the server was able to locate the entry referred to by the baseObject but was unable to search all the entries in the scope at and under the baseObject, the server may return one or more SearchResultReference, each containing a reference to another set of servers for continuing the operation. A server should not return any SearchResultReference if it has not located the baseObject and thus has not searched any entries; in this case it would return a SearchResultDone containing a referral resultCode.

In the absence of indexing information provided to a server from servers holding subordinate naming contexts, SearchResultReference responses are not affected by search filters and are always returned when in scope.

6. **Modify Operation** – this function enables the client to request that a directory entry be modified by the server on its behalf. It is defined as:

```
ModifyRequest::=[APPLICATION6]SEQUENCE{
   object              DAPDN,
   modification    SEQUENCE OF SEQUENCE {
   operation ENUMERATED {
      add        (0),
      delete     (1),
      replace    (2) },
modification AttributeTypeAndValues } }

AttributeTypeAndValues ::= SEQUENCE {
   type              AttributeDescription,
   vals              SET OF AttributeValue }
```

The parameters of Modify Request are:

object – the object to be modified. The value of this field contains the DN of the entry to be modified. The server will not perform any alias dereferencing in determining the object to be modified.

modification – a list of modifications to be performed on the entry. The entire list of entry modifications must be performed in the order they are listed, as a single atomic operation. While individual modifications may violate the directory schema, the resulting entry after the entire list of modifications is performed should conform to the requirements of the directory schema. The values

that may be taken on by the 'operation' field in each modification construct have the following semantics respectively:

add – this parameter adds values listed to the given attribute, creating the attribute if necessary.

delete – delete values listed from the given attribute, removing the entire attribute if no values are listed, or if all current values of the attribute are listed for deletion.

replace – replace all existing values of the given attribute with the new values listed, creating the attribute if it did not already exist. A replace with no value will delete the entire attribute if it exists, and is ignored if the attribute does not exist.

The LDAP server will return to the client a single Modify Response indicating either the successful completion of the DIT modification, or the reason that the modification failed. Due to the requirement for atomicity in applying the list of modifications in the Modify Request, the client may expect that no modifications of the DIT have been performed if the Modify Response received indicates any sort of error, and that all requested modifications have been performed if the Modify Response indicates successful completion of the Modify Operation. If the connection between the client and the server fails, whether the modification occurred or not is indeterminate.

Please take note that the Modify Operation cannot be used to remove from an entry any of its distinguished values (i.e. values which form the entry's relative distinguished name). An attempt to do so will result in the server returning the error *notAllowedOnRDN*.

7. **Add Operation** – this function allows the LDAP client to request the addition of entry into the directory. It is defined as follows:

```
AddRequest ::= [APPLICATION 8] SEQUENCE {
    entry           LDAPDN,
    attributes      AttributeList }

AttributeList ::= SEQUENCE OF SEQUENCE{
    type            AttributeDescription,
    vals            SET OF AttributeValue }
```

The parameters of the Add Request are:

entry – the Distinguished Name of the entry to be added. Note that the LDAP server will not dereference any aliases in locating the entry to be added.

attributes – the list of attributes that make up the content of the entry being added. Clients LDAP should include distinguished values (those forming the entry's own RDN) in this list, the objectClass attribute, and values of any mandatory attributes of the listed object classes. Clients must not supply the createTimestamp or creatorsName attributes, since these will be generated automatically by the server.

Upon receipt of an Add Request, a server will attempt to perform the add requested. The result of the add attempt will be returned to the client in the Add Response. A response of success indicates that the new entry is present in the directory.

8. **Delete Operation** – this operation allows the client to request the server to remove an entry from the directory. It is defined as follows:

```
DelRequest ::= [APPLICATION 10] LDAPDN
```

The Delete Request consists of the Distinguished Name of the entry to be deleted. Note that the server will not dereference aliases while resolving the name of the target entry to be removed, and that only leaf entries (those with no subordinate entries) can be deleted with this operation.

9. **Modify DN Operation** – this allows the client to modify the leftmost (least important) component of the name of an entry in the directory, or move a subtree of entries to a new location in the same directory. It is defined as follows:

```
ModifyDNRequest ::= [APPLICATION
                     SEQUENCE {
    entry           LDAPDN,
    newrdn          RelativeLDAPDN,
    deleteoldrdn    BOOLEAN,
    newSuperior     [0] LDAPDN OPTIONAL }
```

The parameters of Modify DN Request are:

entry – the Distinguished Name of the entry to be changed. This entry may or may not have subordinate entries.

newrdn – the RDN that will form the leftmost component of the new name of the entry.

deleteoldrdn – a boolean parameter that controls whether the old RDN attribute values are to be retained as attributes of the entry, or deleted from the entry.

newSuperior – this parameter becomes the Distinguished Name of the entry which becomes the

immediate superior of the existing entry.

After receiving a ModifyDNRequest, the LDAPserver will try to perform the name change. The result of the name change attempt will be returned to the LDAP client in the Modify DN Response.

10. **Compare Operation** – this function enables the LDAP client to compare an assertion with an entry in the directory. The Compare Request is defined as follows:

```
CompareRequest::=[APPLICATION14]
                    SEQUENCE{
entry                 LDAPDN,
ava                   AttributeValueAssertion }
```

Upon receipt of a Compare Request, the LDAP server will attempt to perform the requested comparison. The result of the comparison will be returned to the client in the Compare Response. Note that errors and the result of comparison are all returned in the same construct .

11. **Abandon Operation** – this allows the LDAP client to request that the server abandon an outstanding operation. The Abandon Request is defined as follows:

```
AbandonRequest::= [APPLICATION 16]
                    MessageID
```

The MessageID must be that of an operation which was requested earlier in this connection. (The abandon request itself has its own message id. This is distinct from the id of the earlier operation being abandoned.)

There is no response defined in the Abandon Operation. Upon transmission of an Abandon Operation, an LDAP client may expect that the operation identified by the Message ID in the Abandon Request has been abandoned. In the event that the LDAP server receives an Abandon Request from the client on a Search Operation in the midst of transmitting responses to the search, that server must cease transmitting entry responses to the abandoned request immediately, and must not send the SearchResponseDone. Of course, the server should make sure that only properly encoded LDAPMessage PDU's are transmitted.

12. **Extended Operation** – LDAP3 includes an extension mechanism to allow additional operations to be defined for services not supported by the protocol (i.e. digitally signed operations and results).

The Extended Operation allows clients to make requests and receive responses with predefined syntaxes and semantics. These may be defined in RFCs or be private to particular implementations. Each request must have a unique object identifier assigned to it. For example:

```
ExtendedRequest::= [APPLICATION23]
                        SEQUENCE {
  requestName [0] LDAPOID,
  requestValue [1] OCTET STRING OPTIONAL }
```

The requestName is a dotted-decimal representation of the object identifier that corresponds to the request. The requestValue is information in a form defined by that request, encapsulated inside an octet string.

LDAP Element Encodings and Transfer

The various elements of LDAP are encoded for transmission using the BER of ASN.1. But since certain elements of BER require high processing overhead, the following additional restrictions are imposed on BER encodings of the elements:

- Only the definite form of length encoding will be used.
- Octet string values will be encoded in the primitive form only.
- If the value of a Boolean type is true, the encoding must have its content set to hex “FF”.
- If a value of a type is its default value, it must be absent.

Note: These restrictions are not applied to ASN.1 types encapsulated inside octet string values.

Security Issues

LDAP3 supports the authentication mechanisms defined for LDAP2 (simple authentication using a cleartext password and Kerberos version 4.0), as well as any SASL mechanism. However, the use of cleartext password in LDAP is strongly discouraged, particularly when the transport service cannot guarantee confidentiality.

Among the most common SASL mechanisms used in LDAP3 are the Secure Socket Layer (SSL) protocol and the Transport Layer Security (TLS) protocol. Both protocols support server authentication, client authentication, or mutual authentication. Another popular authentication mechanism is Kerberos, which relies on the Data Encryption Standard (DES) to encrypt messages. It is a third-party authentication scheme that provides security for the authentication process using a separate server. LDAP2 can also be

configured to run over the SSL/TLS transport layer.

Applications that cache attributes and entries obtained via LDAP must ensure that access controls are maintained if that information is to be provided to multiple clients, since servers may have access control policies which prevent the return of entries or attributes in search results except to particular authenticated clients. For example, caches could serve result information only to the client whose request caused it to be cached.

Synchronizing LDAP Directories

In an X.500 directory, replication of information among the DSA's is carried out via the DISP. This process enables each DSA's DIB to keep a shadow or replicated copy of the information stored in the directory system. A shadowed copy is a read-only copy while a replicated copy can be accessed for bilateral updates.

To enable LDAP directories to support replication and synchronization without using the DISP, the University of Michigan developed the Standalone LDAP Update Replication Daemon (slurpd) to work with its Standalone LDAP Daemon (slapd). In an LDAP directory, DSA's are arranged based on a master-slave relationship. A master server can be assigned to each administrative division in a company or in an organization, with all servers located lower in the hierarchy being considered as slave servers. In this setup, master directory servers are given the capability to update the subtree's directories found in the slave servers. Slurpd is a replication agent that detects the changes made to a master directory server and propagates those changes to slave slapd servers via the LDAP. It reads the replication log produced by the master slapd to detect if information has been updated. If the replication log file is empty or does not exist, slurpd goes to sleep, awaking periodically to check if there are changes to directory information that must be relayed to the slave servers. A special code in the slave servers ensures that the create and modify timestamps are recorded unchanged when the slaves receive replicated changes.

An example of how the master-slave replication scheme works is shown in Figure 5. In the figure, the LDAP client forward a requests to modify the directory

continued on page 15

CASE STUDY: Sending and receiving

continued from page 1

Why Internet Exchange?

Gilbert learned about Internet Exchange via the World Wide Web, and since installing the gateway, sending and receiving email via the Internet has become hassle-free. And he has also taken down the SoftSwitch gateway, relying solely on Internet Exchange for all internal cc:Mail connectivity. At present, the Indiana State Government has five different local mail platforms supported by five different gateways, four of which are non-UNIX.

"Internet Exchange is a solid and robust product that has been serving our cc:Mail community for several years now. It triggered, within our organization, a statewide transformation to a modern intranet environment based on Internet standards," says Gilbert. "We're looking forward for new releases (from IMA), and hope that IMA continues to listen to its customers for new features that can be incorporated in future versions of Internet Exchange."

Gilbert adds that Internet Exchange's user-friendliness becomes more apparent now that they are migrating email users from cc:Mail to either GroupWise or MS Exchange. "We hope to continue using Internet Exchange as a migration tool in our SMTP infrastructure until we can finally migrate all of our users off cc:Mail permanently. We expect our number of cc:Mail users to drop below 1,000 within the next two to three years."

Gilbert is also very impressed with the IMA mailing list. "The IMA mailing list has also been quite useful in allowing users air their problems or discuss issues related to IMA. I myself have posted messages on several occasions."

And his overall impression of Internet Exchange? "I have to admit, out of all the non-UNIX SMTP gateways we have here, IMA's gateway has been in just every aspect the best one," Gilbert concludes.

Internet Exchange News

A monthly publication of International Messaging Associates, Ltd.

Staff

Editors ..... Rommel Fajardo, Taffi Zaidi
Editorial Consultant ..... Tim Kehres
Graphic Artists ..... Rommel Fajardo, Taffi Zaidi
Contributors..... Jennifer Villamor, Imee Villeza

Please send your comments and suggestions to: doc@ima.com

Questions & Answers

Q: Why do I receive mime*.raw attachments?

A: Internet Exchange assigns a message attachment a name in the form of "mime<n>.raw" in the event that the MIME headers for the said attachment lack a "name=" parameter for the *Content-Type* header or a "filename=" parameter for the *Content-Disposition* header. In the absence of either of these parameters, Internet Exchange has no way of knowing what to call the attachment, so the gateway uses a generic name and file type. If the MIME content type and subtype match an entry in the gateway's list of accepted MIME mappings, the appropriate application will then assign the corresponding suffix to the generic name (i.e. mime<n>.doc in the case of MS Word).

Q: How many entries can I put in the *Alternate host/domain names* field in the *Configure Connection* screen? In my experience with Internet Exchange version 2.12, this field refused to accept additional names after only so many entries.

A: Older versions of Internet Exchange are designed to have an internal buffer storage that can handle a maximum of 2K of alternate host/domain names only. Internet Exchange 3.11 has been designed with an internal buffer storage size of 8K. Thus, the *Alternate host/domain names* field is definitely capable of handling more entries in the latest version.

Q: We are using Internet Exchange 2.11 for cc:Mail. I noticed that each time one of our users sends a message to more than 40 recipients, he gets the message "Message not deliverable" on his screen. I checked the ieccmail logfile and found this:

Fri Oct 09 15:37:14 smtpd: 349> 421 local limit of 40 sessions exceeded. Please call back later.

How can I configure Internet Exchange to exceed the limit of 40 SMTPD sessions?

A: Please upgrade your gateway to Internet Exchange 3.11 since version 2.x is designed to handle a maximum of only 40 concurrent SMTPD connections.

Internet Exchange 3.11 allows 256 concurrent SMTPD connections and 120 concurrent SMTPC connections. SMTPD is a server process that continuously runs on the gateway machine and listens for incoming SMTP connections on TCP port 25. SMTPC is the client program that establishes the required number of connections with external SMTP servers and transfers outbound messages to the appropriate mail hosts.

To upgrade to Internet Exchange v3.11, go to http://www.ima.com. Those sites using current v3.0 and are current with their support/maintenance status are eligible for a free upgrade to version 3.11. Those using version 2.x can visit IMA's website at http://www.ima.com for information on how to upgrade to version 3.11

"This 'telephone' has too many shortcomings to be seriously considered as a means of communication. The device is inherently of no value to us." -- Western Union internal memo, 1876.

This Month's Tips

The Fully Qualified Domain Name

When upgrading your gateway to a later version of Internet Exchange, make sure that you enter the fully qualified domain name (FQDN) of your machine in the space provided for together with the expiry date (for interim licenses only), serial number, and license key. For example, if your host name is *manila* and the domain name is *ima.com*, your FQDN is *manila.ima.com* (not manila@ima.com).

Upgrading Internet Exchange 3.0 to 3.11

To upgrade your gateway from v3.0 to v3.11, go to http://www.ima.com/product/ccmail/3.1/update.html and enter the FQDN, license number, and expiry date (for interim licenses only) in the space provided. Your are then ready to download *c311upd.exe*. Do not save this file in the ieccmail directory. It is advised that a new folder be created for storing *c311upd.exe*. Then unzip *c311upd.exe* and look for *1update.exe*. Running this program will automatically upgrade your gateway from v3.0 to v3.11. Those sites using v3.0 and are current with their support status are eligible for a free upgrade to v3.11.

continued from page 13

(a). The slave directory then forwards the client's request to the master directory using the LDAP (b). After making the changes requested by the client, the master directory forwards these changes to the slave directories to keep the directory contents synchronized (c).

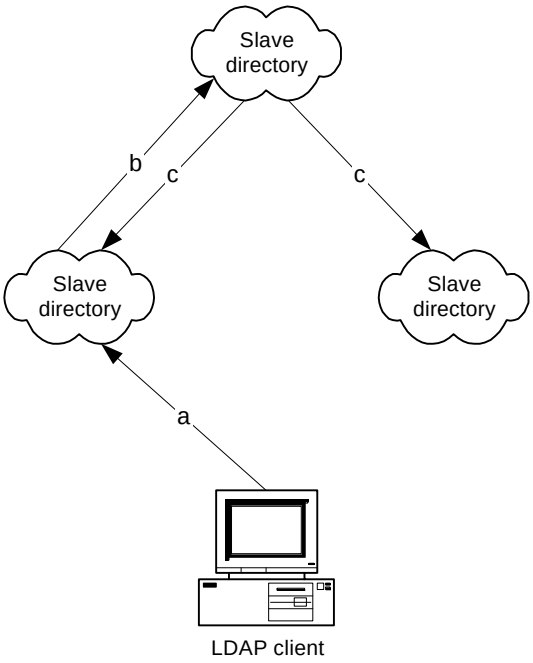

Figure 5. Master-slave replication scheme

Another replication technique that can be used for directory synchronization is the multi-master replication method. This approach is based on a replication model where entries can be written and updated on several slave servers without requiring prior communication with other master servers.

Aside from the solution provided by the University of Michigan's slurpd, there are many products on the market that allows replication and synchronization of information located in distributed directory systems. Among such products are Sun Microsystems' Sun Directory Services 1.0, Netscape's NT Directory Synchronization Service, Netvision's Synchronicity, Innosoft International Inc.'s Innosoft Directory Service, Enterprise Solutions Ltd.'s EXM/SyncWare 2.0, and Digital Equipment Corp.'s LDAP Directory Synchronization Utility (LDSU). These products are designed to support proprietary LDAP directories such as the Sun Internet Mail Server and Netscape Directory Server. Recently, the LDSU was used by Digital Equipment and Compaq to come up with a unified directory within minutes after the two companies announced a merger.

Another emerging synchronization technology for LDAP servers is the use of metadirectories. One such product is Zoomit Corp.'s Via metadirectory, which connects different directories together by supporting several protocols, including the Dynamic Host Configuration Protocol, SMTP, and HTTP. Another is IBM's Directory and Security Server (DSS). The DSS includes an X.500 directory store and supports cross-directory synchronization. It uses the Soft-Switch Directories from Lotus Development Corp. as the metadirectory engine. Metadirectories gather data from different directories using software agents and store it in a unified repository.

Conclusion

The LDAP provides the Internet community with a very useful tool for managing information in a directory service over a TCP connection with minimal processing overhead. It makes electronic communication a lot easier by providing us with an online system for locating organizations, individuals, files, and devices in a network, whether on an intranet or on the Internet. The LDAP Bind operation, which establishes connection between the server and the client, for example, also supports a choice of credentials, allowing new (and hopefully more secure) authentication methods to be included in the future. In addition, the industry is in the process of defining new standards that will add new and useful features to the LDAP.

Microsoft Corp. and Netscape Communications Corp., for instance, are involved in collaborative research aimed at developing new API's for LDAP. These two companies, together with Novell, are also proposing replication mechanisms to the Internet Engineering Task Force (IETF). In addition, the IETF has already published an Internet draft (which may soon be classified into an RFC) that identifies the fundamental requirements for the replication of data accessible via the LDAP. This document, titled "LDAP Replication Requirements," describes a replication model describing an LDAP replication scheme that will provide interoperability among directory services from different vendors. The document also describes a replication protocol that will support incremental synchronization as well as multi-master and master-slave replication.

All these efforts are aimed at providing the Internet (and intranet) community with a truly global protocol that will make the task of finding someone or something on the Internet really very simple.