Lightweight Directory Access Protocol

The **Lightweight Directory Access Protocol** (**LDAP** /ˈɛldæp/) is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network.[1] Directory services play an important role in developing intranet and Internet applications by allowing the sharing of information about users, systems, networks, services, and applications throughout the network.[2] As examples, directory services may provide any organized set of records, often with a hierarchical structure, such as a corporate email directory. Similarly, a telephone directory is a list of subscribers with an address and a phone number.

Lightweight Directory Access Protocol

Communication protocol	
Purpose	Directory service
Based on	X.500
OSI layer	Application layer
Port(s)	389 (ldap), 636 (ldaps)
RFC(s)	RFC 4510, RFC 4511

LDAP is specified in a series of Internet Engineering Task Force (IETF) Standard Track publications called Request for Comments (RFCs), using the description language ASN.1. The latest specification is Version 3, published as RFC 4511 (https://datatracker.ietf.org/doc/html/rfc4511)[3] (a road map to the technical specifications is provided by RFC4510 (https://tools.ietf.org/html/rfc4510)).

A common use of LDAP is to provide a central place to store usernames and passwords. This allows many different applications and services to connect to the LDAP server to validate users.[4]

LDAP is based on a simpler subset of the standards contained within the X.500 standard. Because of this relationship, LDAP is sometimes called X.500-lite.[5]

History

Telecommunication companies' understanding of directory requirements were well developed after some 70 years of producing and managing telephone directories. These companies introduced the concept of directory services to information technology and computer networking, their input culminating in the comprehensive X.500 specification,[6] a suite of protocols produced by the International Telecommunication Union (ITU) in the 1980s.

X.500 directory services were traditionally accessed via the X.500 Directory Access Protocol (DAP), which required the Open Systems Interconnection (OSI) protocol stack. LDAP was originally intended to be a lightweight alternative protocol for accessing X.500 directory services through the simpler (and now widespread) TCP/IP protocol stack. This model of directory access was borrowed from the DIXIE and Directory Assistance Service protocols.

The protocol was originally created[7] by Tim Howes of the University of Michigan, Steve Kille of Isode Limited, Colin Robbins of Nexor and Wengyik Yeong of Performance Systems International, circa 1993, as a successor[8] to DIXIE and DAS. Mark Wahl of Critical Angle Inc., Tim Howes, and Steve Kille started work in 1996 on a new version of LDAP, LDAPv3, under the aegis of the Internet Engineering Task Force (IETF). LDAPv3, first published in 1997, superseded LDAPv2 and added support for extensibility, integrated the Simple Authentication and Security Layer, and better aligned the protocol to the 1993 edition of X.500. Further development of the LDAPv3 specifications themselves and of numerous extensions adding features to LDAPv3 has come through the IETF.

In the early engineering stages of LDAP, it was known as *Lightweight Directory Browsing Protocol*, or *LDBP*. It was renamed with the expansion of the scope of the protocol beyond directory browsing and searching, to include directory update functions. It was given its *Lightweight* name because it was not as network intensive as its DAP predecessor and thus was more easily implemented over the Internet due to its relatively modest bandwidth usage.

LDAP has influenced subsequent Internet protocols, including later versions of X.500, XML Enabled Directory (XED), Directory Service Markup Language (DSML), Service Provisioning Markup Language (SPML), and the Service Location Protocol (SLP). It is also used as the basis for Microsoft's Active Directory.

Protocol overview

A client starts an LDAP session by connecting to an LDAP server, called a Directory System Agent (DSA), by default on TCP and UDP port 389, or on port 636 for LDAPS (LDAP over TLS/SSL, see below).[9] The client then sends an operation request to the server, and a server sends responses in return. With some exceptions, the client does not need to wait for a response before sending the next request, and the server may send the responses in any order. All information is transmitted using Basic Encoding Rules (BER).

The client may request the following operations:

- StartTLS – use the LDAPv3 Transport Layer Security (TLS) extension for a secure connection
- Bind – authenticate and specify LDAP protocol version
- Search – search for and/or retrieve directory entries
- Compare – test if a named entry contains a given attribute value
- Add a new entry
- Delete an entry
- Modify an entry
- Modify Distinguished Name (DN) – move or rename an entry
- Abandon – abort a previous request
- Extended Operation – generic operation used to define other operations
- Unbind – close the connection (not the inverse of Bind)

In addition the server may send "Unsolicited Notifications" that are not responses to any request, e.g. before the connection is timed out.

A common alternative method of securing LDAP communication is using an SSL tunnel. The default port for LDAP over SSL is 636. The use of LDAP over SSL was common in LDAP Version 2 (LDAPv2) but it was never standardized in any formal specification. This usage has been deprecated along with LDAPv2, which was officially retired in 2003.[10]

Directory structure

The protocol provides an interface with directories that follow the 1993 edition of the X.500 model:

- An entry consists of a set of attributes.
- An attribute has a name (an *attribute type* or *attribute description*) and one or more values. The attributes are defined in a *schema* (see below).
- Each entry has a unique identifier: its *Distinguished Name* (DN). This consists of its *Relative Distinguished Name* (RDN), constructed from some attribute(s) in the entry, followed by the parent entry's DN. Think of the DN as the full file path and the RDN as its relative filename in its parent folder (e.g. if `/foo/bar/myfile.txt` were the DN, then `myfile.txt` would be the RDN).

A DN may change over the lifetime of the entry, for instance, when entries are moved within a tree. To reliably and unambiguously identify entries, a UUID might be provided in the set of the entry's *operational attributes*.

An entry can look like this when represented in LDAP Data Interchange Format (LDIF), a plain text format (as opposed a binary protocol such as LDAP itself):

```
dn: cn=John Doe,dc=example,dc=com
cn: John Doe
givenName: John
sn: Doe
telephoneNumber: +1 888 555 6789
telephoneNumber: +1 888 555 1232
mail: john@example.com
manager: cn=Barbara Doe,dc=example,dc=com
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
```

"`dn`" is the distinguished name of the entry; it is neither an attribute nor a part of the entry. "`cn=John Doe`" is the entry's RDN (Relative Distinguished Name), and "`dc=example,dc=com`" is the DN of the parent entry, where "`dc`" denotes 'Domain Component'. The other lines show the attributes in the entry. Attribute names are typically mnemonic strings, like "`cn`" for common name, "`dc`" for domain component, "`mail`" for email address, and "`sn`" for surname.

A server holds a subtree starting from a specific entry, e.g. "`dc=example,dc=com`" and its children. Servers may also hold references to other servers, so an attempt to access "`ou=department,dc=example,dc=com`" could return a *referral* or *continuation reference* to a server that holds that part of the directory tree. The client can then contact the other server. Some servers also support *chaining*, which means the server contacts the other server and returns the results to the client.

LDAP rarely defines any ordering: The server may return the values of an attribute, the attributes in an entry, and the entries found by a search operation in any order. This follows from the formal definitions - an entry is defined as a set of attributes, and an attribute is a set of values, and sets need not be ordered.

Operations

Add

The ADD operation inserts a new entry into the directory-server database.[11] If the distinguished name in the add request already exists in the directory, then the server will not add a duplicate entry but will set the result code in the add result to decimal 68, "entryAlreadyExists".[12]

- LDAP-compliant servers will never dereference the distinguished name transmitted in the add request when attempting to locate the entry, that is, distinguished names are never de-aliased.
- LDAP-compliant servers will ensure that the distinguished name and all attributes conform to naming standards.
- The entry to be added must not exist, and the immediate superior must exist.

```
dn: uid=user,ou=people,dc=example,dc=com
changetype: add
objectClass:top
objectClass:person
uid: user
sn: last-name
cn: common-name
userPassword: password
```

In the above example, `uid=user,ou=people,dc=example,dc=com` must not exist, and `ou=people,dc=example,dc=com` must exist.

Bind (authenticate)

When an LDAP session is created, that is, when an LDAP client connects to the server, the **authentication state** of the session is set to anonymous. The BIND operation establishes the authentication state for a session.

Simple BIND and SASL PLAIN can send the user's DN and password in plaintext, so the connections utilizing either Simple or SASL PLAIN should be encrypted using Transport Layer Security (TLS). The server typically checks the password against the `userPassword` attribute in the named entry. Anonymous BIND (with empty DN and password) resets the connection to anonymous state.

SASL (Simple Authentication and Security Layer) BIND provides authentication services through a wide range of mechanisms, e.g. Kerberos or the client certificate sent with TLS.[13]

BIND also sets the LDAP protocol version by sending a version number in the form of an integer. If the client requests a version that the server does not support, the server must set the result code in the BIND response to the code for a protocol error. Normally clients should use LDAPv3, which is the default in the protocol but not always in LDAP libraries.

BIND had to be the first operation in a session in LDAPv2, but is not required as of LDAPv3. In LDAPv3, each successful BIND request changes the authentication state of the session and each unsuccessful BIND request resets the authentication state of the session.

Delete

To delete an entry, an LDAP client transmits a properly formed delete request to the server.[14]

- A delete request must contain the distinguished name of the entry to be deleted
- Request controls may also be attached to the delete request
- Servers do not dereference aliases when processing a delete request
- Only leaf entries (entries with no subordinates) may be deleted by a delete request. Some servers support an operational attribute `hasSubordinates` whose value indicates whether an entry has any subordinate entries, and some servers support an operational

attribute `numSubordinates`[15] indicating the number of entries subordinate to the entry containing the `numSubordinates` attribute.
- Some servers support the subtree delete request control permitting deletion of the DN and all objects subordinate to the DN, subject to access controls. Delete requests are subject to access controls, that is, whether a connection with a given authentication state will be permitted to delete a given entry is governed by server-specific access control mechanisms.

Search and compare

The Search operation is used to both search for and read entries. Its parameters are:

baseObject
: The name of the base object entry (or possibly the root) relative to which the search is to be performed.

scope
: What elements below the baseObject to search. This can be `BaseObject` (search just the named entry, typically used to read one entry), `singleLevel` (entries immediately below the base DN), or `wholeSubtree` (the entire subtree starting at the base DN).

filter
: Criteria to use in selecting elements within scope. For example, the filter `(&(objectClass=person)(|(givenName=John)(mail=john*)))` will select "persons" (elements of objectClass `person`) where the matching rules for `givenName` and `mail` determine whether the values for those attributes match the filter assertion. Note that a common misconception is that LDAP data is case-sensitive, whereas in fact matching rules and ordering rules determine matching, comparisons, and relative value relationships. If the example filters were required to match the case of the attribute value, an *extensible match filter* must be used, for example, `(&(objectClass=person)(|(givenName:caseExactMatch:=John)(mail:caseExactSubstringsMatch:=john*)))`

derefAliases
: Whether and how to follow alias entries (entries that refer to other entries),

attributes
: Which attributes to return in result entries.

sizeLimit, timeLimit
: Maximum number of entries to return, and maximum time to allow search to run. These values, however, cannot override any restrictions the server places on size limit and time limit.

typesOnly
: Return attribute types only, not attribute values.

The server returns the matching entries and potentially continuation references. These may be returned in any order. The final result will include the result code.

The Compare operation takes a DN, an attribute name and an attribute value, and checks if the named entry contains that attribute with that value.

Modify

The MODIFY operation is used by LDAP clients to request that the LDAP server make changes to existing entries.[16] Attempts to modify entries that do not exist will fail. MODIFY requests are subject to access controls as implemented by the server.

The MODIFY operation requires that the distinguished name (DN) of the entry be specified, and a sequence of changes. Each change in the sequence must be one of:

- add (add a new value, which must not already exist in the attribute)
- delete (delete an existing value)
- replace (replace an existing value with a new value)

LDIF example of adding a value to an attribute:

```
dn: dc=example,dc=com
changetype: modify
add: cn
cn: the-new-cn-value-to-be-added
-
```

To replace the value of an existing attribute, use the `replace` keyword. If the attribute is multi-valued, the client must specify the value of the attribute to update.

To delete an attribute from an entry, use the keyword `delete` and the changetype designator `modify`. If the attribute is multi-valued, the client must specify the value of the attribute to delete.

There is also a Modify-Increment extension[17] which allows an incrementable attribute value to be incremented by a specified amount. The following example using LDIF increments `employeeNumber` by `5`:

```
dn: uid=user.0,ou=people,dc=example,dc=com
changetype: modify
increment: employeeNumber
employeeNumber: 5
-
```

When LDAP servers are in a replicated topology, LDAP clients should consider using the post-read control to verify updates instead of a search after an update.[18] The post-read control is designed so that applications need not issue a search request after an update – it is bad form to retrieve an entry for the sole purpose of checking that an update worked because of the replication eventual consistency model. An LDAP client should not assume that it connects to the same directory server for each request because architects may have placed load-balancers or LDAP proxies or both between LDAP clients and servers.

Modify DN

Modify DN (move/rename entry) takes the new RDN (Relative Distinguished Name), optionally the new parent's DN, and a flag that indicates whether to delete the value(s) in the entry that match the old RDN. The server may support renaming of entire directory subtrees.

An update operation is atomic: Other operations will see either the new entry or the old one. On the other hand, LDAP does not define transactions of multiple operations: If you read an entry and then modify it, another client may have updated the entry in the meantime. Servers may implement extensions[19] that support this, though.

Extended operations

The Extended Operation is a generic LDAP operation that can define new operations that were not part of the original protocol specification. StartTLS is one of the most significant extensions. Other examples include Cancel and Password Modify.

StartTLS

The StartTLS operation establishes Transport Layer Security (the descendant of SSL) on the connection. It can provide data confidentiality (to protect data from being observed by third parties) and/or data integrity protection (which protects the data from tampering). During TLS negotiation the server sends its X.509 certificate to prove its identity. The client may also send a certificate to prove its identity. After doing so, the client may then use SASL/EXTERNAL. By using the SASL/EXTERNAL, the client requests the server derive its identity from credentials provided at a lower level (such as TLS). Though technically the server may use any identity information established at any lower level, typically the server will use the identity information established by TLS.

Servers also often support the non-standard "LDAPS" ("Secure LDAP", commonly known as "LDAP over SSL") protocol on a separate port, by default 636. LDAPS differs from LDAP in two ways: 1) upon connect, the client and server establish TLS before any LDAP messages are transferred (without a StartTLS operation) and 2) the LDAPS connection must be closed upon TLS closure.

Some "LDAPS" client libraries only encrypt communication; they do not check the host name against the name in the supplied certificate.[20]

Abandon

The Abandon operation requests that the server abort an operation named by a message ID. The server need not honor the request. Neither Abandon nor a successfully abandoned operation send a response. A similar Cancel extended operation does send responses, but not all implementations support this.

Unbind

The Unbind operation abandons any outstanding operations and closes the connection. It has no response. The name is of historical origin, and is *not* the opposite of the Bind operation.[21]

Clients can abort a session by simply closing the connection, but they should use Unbind.[22] Unbind allows the server to gracefully close the connection and free resources that it would otherwise keep for some time until discovering the client had abandoned the connection. It also instructs the server to cancel operations that can be canceled, and to not send responses for operations that cannot be canceled.[23]

URI scheme

An LDAP uniform resource identifier (URI) scheme exists, which clients support in varying degrees, and servers return in referrals and continuation references (see RFC 4516):

```
ldap://host:port/DN?attributes?scope?filter?extensions
```

Most of the components described below are optional.

- *host* is the FQDN or IP address of the LDAP server to search.

- *port* is the network port (default port 389) of the LDAP server.
- *DN* is the distinguished name to use as the search base.
- *attributes* is a comma-separated list of attributes to retrieve.
- *scope* specifies the search scope and can be "base" (the default), "one" or "sub".
- *filter* is a search filter. For example, `(objectClass=*)` as defined in RFC 4515.
- *extensions* are extensions to the LDAP URL format.

For example, "`ldap://ldap.example.com/cn=John%20Doe,dc=example,dc=com`" refers to all user attributes in John Doe's entry in `ldap.example.com`, while "`ldap:///dc=example,dc=com??sub?(givenName=John)`" searches for the entry in the default server (note the triple slash, omitting the host, and the double question mark, omitting the attributes). As in other URLs, special characters must be percent-encoded.

There is a similar non-standard `ldaps` URI scheme for LDAP over SSL. This should not be confused with LDAP with TLS, which is achieved using the StartTLS operation using the standard `ldap` scheme.

Schema

The contents of the entries in a subtree are governed by a directory schema, a set of definitions and constraints concerning the structure of the directory information tree (DIT).

The schema of a Directory Server defines a set of rules that govern the kinds of information that the server can hold. It has a number of elements, including:

- Attribute Syntaxes—Provide information about the kind of information that can be stored in an attribute.
- Matching Rules—Provide information about how to make comparisons against attribute values.
- Matching Rule Uses—Indicate which attribute types may be used in conjunction with a particular matching rule.
- Attribute Types—Define an object identifier (OID) and a set of names that may refer to a given attribute, and associates that attribute with a syntax and set of matching rules.
- Object Classes—Define named collections of attributes and classify them into sets of required and optional attributes.
- Name Forms—Define rules for the set of attributes that should be included in the RDN for an entry.
- Content Rules—Define additional constraints about the object classes and attributes that may be used in conjunction with an entry.
- Structure Rule—Define rules that govern the kinds of subordinate entries that a given entry may have.

Attributes are the elements responsible for storing information in a directory, and the schema defines the rules for which attributes may be used in an entry, the kinds of values that those attributes may have, and how clients may interact with those values.

Clients may learn about the schema elements that the server supports by retrieving an appropriate subschema subentry.

The schema defines *object classes*. Each entry must have an objectClass attribute, containing named classes defined in the schema. The schema definition of the classes of an entry defines what kind of object the entry may represent - e.g. a person, organization or domain. The object class definitions also define the list of attributes that must contain values and the list of attributes which may contain values.

For example, an entry representing a person might belong to the classes "top" and "person". Membership in the "person" class would require the entry to contain the "sn" and "cn" attributes, and allow the entry also to contain "userPassword", "telephoneNumber", and other attributes. Since entries may have multiple ObjectClasses values, each entry has a complex of optional and mandatory attribute sets formed from the union of the object classes it represents. ObjectClasses can be inherited, and a single entry can have multiple ObjectClasses values that define the available and required attributes of the entry itself. A parallel to the schema of an objectClass is a class definition and an instance in Object-oriented programming, representing LDAP objectClass and LDAP entry, respectively.

Directory servers may publish the directory schema controlling an entry at a base DN given by the entry's subschemaSubentry operational attribute. (An *operational attribute* describes operation of the directory rather than user information and is only returned from a search when it is explicitly requested.)

Server administrators can add additional schema entries in addition to the provided schema elements. A schema for representing individual people within organizations is termed a white pages schema.

Variations

A lot of the server operation is left to the implementor or administrator to decide. Accordingly, servers may be set up to support a wide variety of scenarios.

For example, data storage in the server is not specified - the server may use flat files, databases, or just be a gateway to some other server. Access control is not standardized, though there has been work on it and there are commonly used models. Users' passwords may be stored in their entries or elsewhere. The server may refuse to perform operations when it wishes, and impose various limits.

Most parts of LDAP are extensible. Examples: One can define new operations. *Controls* may modify requests and responses, e.g. to request sorted search results. New search scopes and Bind methods can be defined. Attributes can have *options* that may modify their semantics.

Other data models

As LDAP has gained momentum, vendors have provided it as an access protocol to other services. The implementation then recasts the data to mimic the LDAP/X.500 model, but how closely this model is followed varies. For example, there is software to access SQL databases through LDAP, even though LDAP does not readily lend itself to this.[24] X.500 servers may support LDAP as well.

Similarly, data previously held in other types of data stores are sometimes moved to LDAP directories. For example, Unix user and group information can be stored in LDAP and accessed via PAM and NSS modules. LDAP is often used by other services for authentication and/or authorization (what actions a given already-authenticated user can do on what service). For example in Active Directory Kerberos is used in the authentication step, while LDAP is used in the authorization step.

An example of such data model is the GLUE Schema,[25] which is used in a distributed information system based on LDAP that enable users, applications and services to discover which services exist in a Grid infrastructure and further information about their structure and state.

Usage

An LDAP server may return referrals to other servers for requests that it cannot fulfill itself. This requires a naming structure for LDAP entries so one can find a server holding a given distinguished name (DN), a concept defined in the X.500 Directory and also used in LDAP. Another way of locating LDAP servers for an organization is a DNS server record (SRV).

An organization with the domain example.org may use the top level LDAP DN `dc=example, dc=org` (where *dc* means domain component). If the LDAP server is also named ldap.example.org, the organization's top level LDAP URL becomes `ldap://ldap.example.org/dc=example,dc=org`.

Primarily two common styles of naming are used in both X.500 [2008] and LDAPv3. These are documented in the ITU specifications and IETF RFCs. The original form takes the top level object as the country object, such as `c=US`, `c=FR`. The domain component model uses the model described above. An example of country based naming could be `l=Locality, ou=Some Organizational Unit, o=Some Organization, c=FR`, or in the US: `cn=Common Name, l=Locality, ou=Some Organizational Unit, o=Some Organization, st=CA, c=US`.

See also

- Ambiguous name resolution
- CCSO Nameserver
- Federated Naming Service
- Hesiod (name service)
- Hierarchical database model
- Key server (cryptographic)
- LDAP Application Program Interface
- List of LDAP software
- Simple Authentication and Security Layer (SASL)

References

1. "Network Working Group RFC 4511" (https://tools.ietf.org/rfc/rfc4511.txt). IETF.org. 2006-06-01. Retrieved 2014-04-04.
2. "Directory Services LDAP" (https://docs.oracle.com/cd/A87860_01/doc/ois.817/a83729/adois09.htm). Oracle.com. Retrieved 2014-04-04.
3. What is LDAP? (http://www.gracion.com/server/whatldap.html). Gracion.com. Retrieved on 2013-07-17.
4. "Introduction to OpenLDAP Directory Services" (http://www.openldap.org/doc/admin24/intro.html). *OpenLDAP*. Retrieved 1 February 2016.
5. "LDAP - Lightweight Directory Access Protocol" (http://www.webopedia.com/TERM/L/LDAP.html). Webopedia.com. 4 December 1996. Retrieved 2014-04-05.
6. The X.500 series - ITU-T Rec. X.500 to X.521

7. Howes, Tim. "The Lightweight Directory Access Protocol: X.500 Lite" (http://www.openldap.org/pub/umich/ldap.pdf) (PDF). Retrieved 26 December 2012.
8. "Pre-History of LDAP" (http://cybermatters.info/2013/04/09/prehistory-of-ldap/). *Cyber Matters*. 2013-04-09. Retrieved 5 October 2014.
9. "Service Name and Transport Protocol Port Number Registry" (https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=ldap). IANA. Retrieved 24 March 2021.
10. RFC3494 (http://tools.ietf.org/html/rfc3494)
11. Add section of RFC4511 (http://tools.ietf.org/html/rfc4511#section-4.7)
12. LDAP result codes (http://tools.ietf.org/html/rfc4511#appendix-A)
13. SASL Mechanisms at IANA (https://www.iana.org/assignments/sasl-mechanisms/sasl-mechanisms.xml)
14. RFC4511: delete request (http://tools.ietf.org/html/rfc4511#section-4.8)
15. Boreham Draft (numSubordinates) (http://tools.ietf.org/html/draft-boreham-numsubordinates-01)
16. Modify Section of RFC4511 (http://tools.ietf.org/html/rfc4511#section-4.6)
17. Zeilenga, K. *LDAP Modify-Increment Extension* (https://datatracker.ietf.org/doc/html/rfc4525). doi:10.17487/RFC4525 (https://doi.org/10.17487%2FRFC4525). RFC 4525 (https://datatracker.ietf.org/doc/html/rfc4525).
18. Zeilenga, K. *Lightweight Directory Access Protocol (LDAP) Read Entry Controls* (https://datatracker.ietf.org/doc/html/rfc4527). IETF. doi:10.17487/RFC4527 (https://doi.org/10.17487%2FRFC4527). RFC 4527 (https://datatracker.ietf.org/doc/html/rfc4527).
19. INTERNET-DRAFT LDAP Transactions draft-zeilenga-ldap-txn-15.txt (http://tools.ietf.org/html/draft-zeilenga-ldap-txn-15)
20. Shibboleth Security alert 20120227 (http://shibboleth.net/community/advisories/secadv_20120227.txt)
21. Tools.ietf.org (http://tools.ietf.org/html/rfc4511#section-4.3)
22. Tools.ietf.org (http://tools.ietf.org/html/rfc4511#section-5.3)
23. Tools.ietf.org (http://tools.ietf.org/html/rfc4511#section-3.1)
24. Openldap.org (http://www.openldap.org/doc/admin24/backends.html#SQL)
25. Open Grid Forum : Project Home (https://www.ogf.org/documents/GFD.218.pdf)

Sources

- ITU-T Rec. X.680, "Abstract Syntax Notation One (ASN.1) - Specification of Basic Notation", 1994
- Basic encoding rules (BER) - ITU-T Rec. X.690, "Specification of ASN.1 encoding rules: Basic, Canonical, and Distinguished Encoding Rules", 1994
- RFC 3641 (https://datatracker.ietf.org/doc/html/rfc3641) - Generic String Encoding Rules (GSER) for ASN.1 Types
- RFC 4346 (https://datatracker.ietf.org/doc/html/rfc4346) - The TLS Protocol Version 1.1
- RFC 4422 (https://datatracker.ietf.org/doc/html/rfc4422) - Simple Authentication and Security Layer (SASL)

- SASL mechanisms registered at IANA

Further reading

- Arkills, B (2003). *LDAP Directories Explained: An Introduction and Analysis* (http://www.informit.com/store/ldap-directories-explained-an-introduction-and-analysis-9780201787924). Addison-Wesley Professional. ISBN 978-0-201-78792-4.
- Carter, G (2003). *LDAP System Administration* (https://archive.org/details/ldapsystemadmini00cart). O'Reilly Media. ISBN 978-1-56592-491-8.
- Donley, C (2002). *LDAP Programming, Management, and Integration*. Manning Publications. ISBN 978-1-930110-40-3.
- Howes, T; Smith, M; Good, G (2003). *Understanding and Deploying LDAP Directory Services* (http://www.informit.com/store/understanding-and-deploying-ldap-directory-services-9780672323164). Addison-Wesley Professional. ISBN 978-0-672-32316-4.
- Rhoton, J (1999). *Programmer's Guide to Internet Mail: SMTP, POP, IMAP, and LDAP*. Elsevier. ISBN 978-1-55558-212-8.
- Voglmaier, R (2003). *The ABCs of LDAP: How to Install, Run, and Administer LDAP Services*. Auerbach Publications. ISBN 978-0-8493-1346-2.

External links

- List of public LDAP Servers (2013): "Ldapwiki: Public LDAP Servers" (https://ldapwiki.com/wiki/Public%20LDAP%20Servers). *ldapwiki.com*. 2013. Retrieved 2020-01-18.