excerpt from LDAP RFC 2251 on attributes and attribute options

4.1.4. Attribute Type

An AttributeType takes on as its value the textual string associated with that AttributeType in its specification.

AttributeType ::= LDAPString

Each attribute type has a unique OBJECT IDENTIFIER which has been assigned to it. This identifier may be written as decimal digits with components separated by periods, e.g. "2.5.4.10".

A specification may also assign one or more textual names for an attribute type. These names MUST begin with a letter, and only contain ASCII letters, digit characters and hyphens. They are case insensitive. (These ASCII characters are identical to ISO 10646 characters whose UTF-8 encoding is a single byte between 0x00 and 0x7F.)

If the server has a textual name for an attribute type, it MUST use a textual name for attributes returned in search results. The dotted-decimal OBJECT IDENTIFIER is only used if there is no textual name for an attribute type.

Attribute type textual names are non-unique, as two different specifications (neither in standards track RFCs) may choose the same name.

A server which masters or shadows entries SHOULD list all the attribute types it supports in the subschema entries, using the attributeTypes attribute. Servers which support an open-ended set of attributes SHOULD include at least the attributeTypes value for the 'objectClass' attribute. Clients MAY retrieve the attributeTypes value from subschema entries in order to obtain the OBJECT IDENTIFIER and other information associated with attribute types.

Some attribute type names which are used in this version of LDAP are described in 5. Servers may implement additional attribute types.

4.1.5. Attribute Description

An AttributeDescription is a superset of the definition of the AttributeType. It has the same ASN.1 definition, but allows additional options to be specified. They are also case insensitive.

AttributeDescription ::= LDAPString

A value of AttributeDescription is based on the following BNF:

<AttributeDescription> ::= <AttributeType> [";" <options>]

<options> ::= <option> | <option> ";" <options>

<option> ::= <opt-char> <opt-char>*

<opt-char> ::= ASCII-equivalent letters, numbers and hyphen

Examples of valid AttributeDescription:

cn
userCertificate;binary

One option, "binary", is defined in this document. Additional options may be defined in IETF standards-track and experimental RFCs. Options beginning with "x-" are reserved for private experiments. Any option could be associated with any AttributeType, although not all combinations may be supported by a server.

An AttributeDescription with one or more options is treated as a subtype of the attribute type without any options. Options present in an AttributeDescription are never mutually exclusive. Implementations MUST generate the <options> list sorted in ascending order, and servers MUST treat any two AttributeDescription with the same AttributeType and options as equivalent. A server will treat an AttributeDescription with any options it does not implement as an unrecognized attribute type.

The data type "AttributeDescriptionList" describes a list of 0 or more attribute types. (A list of zero elements has special significance in the Search request.)

AttributeDescriptionList ::= SEQUENCE OF
AttributeDescription

4.1.5.1. Binary Option

If the "binary" option is present in an AttributeDescription, it overrides any string-based encoding representation defined for that attribute in 5. Instead the attribute is to be transferred as a binary value encoded using the Basic Encoding Rules 11. 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.

The presence or absence of the "binary" option only affects the transfer of attribute values in protocol; servers store any particular attribute in a single format. If a client requests that a server return an attribute in the binary format, but the server cannot generate that format, the server MUST treat this attribute type as an unrecognized attribute type. Similarly, clients MUST NOT expect servers to return an attribute in binary format if the client requested that attribute by name without the binary option.

This option is intended to be used with attributes whose syntax is a complex ASN.1 data type, and the structure of values of that type is needed by clients. Examples of this kind of syntax are "Certificate" and "CertificateList".

4.1.6. Attribute Value

A field of type AttributeValue takes on as 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 5.

AttributeValue ::= OCTET STRING

Note that there is no defined limit on the size of this encoding; thus protocol values may include multi-megabyte attributes (e.g. photographs).

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.

Clients MUST NOT send attribute values in a request which are not valid according to the syntax defined for the attributes.

4.1.7. Attribute Value Assertion

The AttributeValueAssertion type definition is similar to the one in the X.500 directory standards. It contains an attribute description and a matching rule assertion value suitable for that type.

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

AssertionValue ::= OCTET STRING
```

If the "binary" option is present in attributeDesc, this signals to the server that the assertionValue is a binary encoding of the assertion value.

For all the string-valued user attributes described in 5, 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. See section 20.2.1.8 of X.501 6 for examples.

4.1.8. Attribute

An attribute consists of a type and one or more values of that type. (Though attributes MUST have at least one value when stored, due to access control restrictions the set may be empty when transferred in protocol. This is described in section 4.5.2, concerning the PartialAttributeList type.)

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

Each attribute value is distinct in the set (no duplicates). The order of attribute values within the vals set is undefined and implementation-dependent, and MUST NOT be relied upon.