Directory Services Markup Language v2.0
December 18, 2001

Approved as an OASIS Standard April 30,2002

1. Introduction

The Directory Services Markup Language v1.0 (DSMLv1) provides a means for representing directory structural information as an XML document.[1]

DSMLv2 goes further, providing a method for expressing directory queries and updates (and the results of these operations) as XML documents. DSMLv2 documents can be used in a variety of ways. For instance, they can be written to files in order to be consumed and produced by programs, or they can be transported over HTTP to and from a server that interprets and generates them.

DSMLv2 functionality is motivated by scenarios including:

- A smart cell phone or PDA needs to access directory information but does not contain an LDAP client.
- A program needs to access a directory through a firewall, but the firewall is not allowed to pass LDAP protocol traffic because it isn't capable of auditing such traffic.
- A programmer is writing an application using XML programming tools and techniques, and the application needs to access a directory.

In short, DSMLv2 is needed to extend the reach of directories.

DSMLv2 is not required to be a strict superset of DSMLv1, which was not designed for upward-compatible extension to meet new requirements. However it is desirable for DSMLv2 to follow the design of DSMLv1 where possible.

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

Errata for this specification can be found at http://oasis-open.org/committees/dsml/errata.

2. Design Approach

DSMLv2 focuses on extending the reach of LDAP directories. Therefore, as in DSMLv1, the design approach is not to abstract the capabilities of LDAP directories

[1] DSMLv1.0 refers to the first DSML specification (http://www.oasis-open.org/committees/dsml/docs/dsml.zip).

as they exist today, but instead to faithfully represent LDAP directories in XML. The difference is that DSMLv1 represented the *state* of a directory while DSMLv2 represents the *operations* that an LDAP directory can perform and the results of such operations.

Therefore the design approach for DSMLv2 is to express LDAP requests and responses as XML document fragments. For the most part DSMLv2 is a systematic translation of LDAP's ASN.1 grammar (defined by RFC 2251) into XML-Schema. Thus, when a DSMLv2 element name matches an identifier in LDAP's ASN.1 grammar, the named element means the same thing in DSMLv2 and in LDAP. Except where noted otherwise, the DSMLv2 grammar follows the same rules as the LDAP grammar, even if those rules are not explicitly expressed in the DSMLv2 schema – for example, a DSMLv2 AttributeDescription can contain only those characters allowed by LDAP.

DSMLv2 is defined in terms of a set of XML fragments that are used as payloads in a binding. A binding defines how the DSMLv2 XML fragments are sent as requests and responses in the context of a specific transport such as SOAP, SMTP, or a simple data file.

DSMLv2 defines two normative bindings: 1) a SOAP request/response binding is defined in section 6; and 2) a file binding that serves as the DSMLv2 analog of LDIF is defined in section 7. The rules for defining other DSMLv2 compliant bindings are found in section 8.

The simple correspondence between LDAP and DSMLv2 has compelling advantages. However there are a few places where it makes sense for DSMLv2 to diverge from LDAP:

1. An LDAP application associates a security principal with an LDAP connection by issuing a Bind request – or, in the SASL Bind case, by issuing as many successive Bind requests as needed to complete the authentication. A DSMLv2 document can be transported via a variety of mechanisms, so the document itself is not used to authenticate the requestor. DSMLv2 includes an Auth request that MAY be used to associate a security principal with a collection of DSMLv2 operations.
2. LDAP does not include a method of grouping operations to be expressed in a single request. DSMLv2 allows multiple LDAP operations to be expressed in one request document, and by specifying a simple positional correspondence between individual requests within a request document and individual responses within a response document. This change is explained in Section 4 below.
3. In LDAP, a single search request typically generates multiple responses, closed by a *searchResultDone* response. To enable the positional correspondence between requests and responses mentioned above, DSMLv2 provides for a binding to wrap the complete set of related search responses into a single *searchResponse* element containing the individual LDAP responses to a search request.
4. The systematic translation of RFC 2251 results in a redundant level of nested element, the *LDAPMessage*. DSMLv2 eliminates this extra level.
5. Defaulting works more naturally in XML documents than in ASN.1 structures, so DSMLv2 uses defaulting in a few places where LDAP doesn't. In DSMLv2 the string-valued elements *matchedDN* and *errorMessage* (from *LDAPResult* in LDAP) and *attributes* (from *SearchRequest* in LDAP) are optional, and when

absent are treated as the empty string. The *sizeLimit*, *timeLimit*, and *typesOnly* elements (from *SearchRequest* in LDAP) default to 0, 0, and FALSE respectively.

In the following the term *provider* is used to refer to that element of the client that implements a binding on behalf of the client. Certain error conditions MAY be detected by the provider without any necessary interaction with a server.

Also note that the term server is used loosely -- clients and programs (not servers) might also perform DSMLv2-based interactions, e.g. in which the client provides a file holding a document as input and the program produces a file containing a document as output. It is clumsy to say "program or server" throughout the document, so we say "server".

3. DSMLv2 URN

The base URN for DSMLv2 is:

```
urn:oasis:names:tc:DSML:2:0:core
```

The above URN is used to construct the URN for the core namespace consisting of the individual operations and responses, a request envelope, a response envelope and an envelope grouping the entries, references and result of a search operation.

Example of using the DSMLv2 URN:

```
<dsml:batchRequest xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core">
```

4. Top-Level Structure

There are two types of DSMLv2 document: the *request* document and the *response* document. In a DSMLv2-based interaction between a client and a server there is a pairing of requests and responses: For each request document submitted by the client there is one response document produced by the server. The structure of the request and response documents depends on the specification of a *binding* of DSMLv2 to some underlying protocol (see sections 6-8 below.)

In the two bindings defined in this specification, the top-level element of a request fragment is a *BatchRequest* and the top-level element of a response fragment is a *BatchResponse*.[2]

A *BatchRequest* contains zero, one, or many individual request elements, while a *BatchResponse* consists of zero, one or many individual response elements. A *BatchRequest* containing zero request elements is a valid request; the valid response is a *BatchResponse* containing zero response elements. Such a batch request-

[2] Note that future bindings may utilize different top-level elements – see section 8.

response pair can be used to verify that a server is capable of processing DSMLv2 documents.

Request-response association

The client and server associate an individual response in a *BatchResponse* with the corresponding individual request in a *BatchRequest* using one (or both) of the following methods: positional correspondence or *RequestID*.

In a positional correspondence, the *n*th response element corresponds to the *n*th request element. For instance, if the third response element is a delete response, then it corresponds to the third request element, a delete request.

When using positional correspondence, a valid batch request-response pair can have fewer responses in the response document than requests in the request document. This can happen due to a syntax error in the request document or due to a failure while processing a request (more on these conditions below). The correspondence stated above still holds: The *n*th response element corresponds to the *n*th request element.

Here is an example of a valid batch request-response pair[3] using positional correspondence:

DSMLv2 Request Document:

```
<batchRequest xmlns="urn:oasis:names:tc:DSML:2:0:core">
  <modifyRequest>…</modifyRequest>
  <addRequest>…</addRequest>
  <delRequest>…</delRequest>
  <addRequest>…</addRequest>
</batchRequest>
```

DSMLv2 Response Document:

```
<batchResponse xmlns="urn:oasis:names:tc:DSML:2:0:core">
  <modifyResponse>…</modifyResponse>
  <addResponse>…</addResponse>
  <delResponse>…</delResponse>
  <addResponse>…</addResponse>
</batchResponse>
```

The alternative to positional correspondence is the use of the optional *requestID* attribute. When the client specifies a value for *requestID* in a request (for example, in an *addRequest*), the server MUST return the same value in the corresponding response (for example, in an *addResponse*). The client need not specify a *requestID* when positional correspondence is also used, although in some cases it may find this useful. For example, when using the file binding for a large file, some clients may find it more convenient to associate failed responses with requests using *requestID* rather than position.

[3] Section 5 below gives details of the contents of request and response elements such as modifyRequest and addResponse.

In bindings (such as those described in this document) where the server generates a *SearchResponse* for a *SearchRequest,* the server MUST associate the *requestID* with the *SearchResponse*, not with its child elements.

Future bindings MAY also use the *requestID* attribute to identify unsolicited notifications. A client MUST NOT send a request with *requestID="*0", as this value is reserved for unsolicited notifications.

A *BatchRequest* element may contain the optional XML-attribute *responseOrder*, which influences how the server orders individual responses within the *BatchResponse*. The valid values are *sequential* and *unordered*. If this attribute is omitted, the default value is *sequential*.

In a *BatchRequest* with *responseOrder*="sequential", the server MUST return a *BatchResponse* in which the individual responses maintain a positional correspondence with the individual requests.

The behavior when *responseOrder*="unordered" is described below in the Parallel Processing section.

Syntax errors

A client may produce a request document that is syntactically incorrect, i.e. does not conform to the XML-Schema for a top-level XML fragment as defined above. In this case the DSMLv2 provider produces a response document to aid in debugging the client. If the server detects the syntax error before performing any directory operations on behalf of the client, the response has the form:

DSMLv2 Response – Syntax error in request:

```
<batchResponse xmlns="urn:oasis:names:tc:DSML:2:0:core">
  <errorResponse type="malformedRequest">
    <message>Unknown element `bogusRequest'  line 87 column 4</message>
  </errorResponse>
</batchResponse>
```

The *errorResponse* element contains details about the error.

If the server performs one or more directory operations on behalf of the client before detecting the syntax error, the server's response contains the response element for each operation that it performed, followed by an *errorResponse* element. For instance,

DSMLv2 Request containing syntax error:

```
<batchRequest xmlns="urn:oasis:names:tc:DSML:2:0:core">
  <modifyRequest>…</modifyRequest>
  <addRequest>…</addRequest>
  <bogusRequest>…</bogusRequest>
  <addRequest>…</addRequest>
   ...
</batchRequest>
```

DSMLv2 Response Document – Syntax error in request:

```
<batchResponse xmlns="urn:oasis:names:tc:DSML:2:0:core">
  <modifyResponse>…</modifyResponse>
  <addResponse>…</addResponse>
  <errorResponse type="malformedRequest">
    <message>Unknown element ‘bogusRequest’  line 87 column 4</message>
  </errorResponse>
</batchResponse>
```

Failures

A client may produce a request document that is syntactically correct but that contains a request that *fails* when the provider executes it. Failure is defined as follows:

- The DSMLv2 provider was unable to connect to a server (represented as an *errorResponse* with *type*="couldNotConnect").
- The DSMLv2 provider connected to a server, but the server closed the connection without responding to the request[4] (represented as an *errorResponse* with *type*="connectionClosed").
- The server returned an *LDAPResultCode* other than 0 ("success"), 6 ("compareTrue"), 5 ("compareFalse"), or 10 ("referral").

When a request execution fails, the server does not attempt to execute later requests within the document. The server produces a response element for each request element that was attempted, including the one that failed.

DSMLv2 Request containing a request that fails

```
<batchRequest xmlns="urn:oasis:names:tc:DSML:2:0:core">
  <modifyRequest>…</modifyRequest>
  <addRequest>…</addRequest>
  <delRequest>…</delRequest>
  <addRequest>…</addRequest>
</batchRequest>
```

DSMLv2 Response – One request not attempted

```
<batchResponse xmlns="urn:oasis:names:tc:DSML:2:0:core">
  <modifyResponse>…</modifyResponse>
  <addResponse>…</addResponse>
  <errorResponse type="connectionClosed"/>
</batchResponse>
```

Parallel processing

A *BatchRequest* element MAY contain the optional XML-attribute *processing*, which influences how the server can process the request elements. The valid values are: *sequential* and *parallel*. If this attribute is omitted, the default value is *sequential*.

[4] Obviously neither this nor the previous condition can happen if the DSMLv2 server is tightly integrated with a directory server, rather than communicating with a directory server via LDAP. On the other hand these conditions may certainly occur within the provider trying to interact with a tightly integrated server.

Example:

```
<batchRequest xmlns="urn:oasis:names:tc:DSML:2:0:core"
                  processing="parallel" >
 …
</batchRequest>
```

In a *BatchRequest* with *processing*=‶sequential″, the server MUST preserve sequential semantics, i.e. it behaves as already described regardless of the value of the *responseOrder* attribute. The effect of processing the *BatchRequest* MUST be as if the request elements were executed in the order they occur within the envelope[5].

In a *BatchRequest* with *processing*=‶parallel″, the server MAY execute the request elements in any order. This form of processing is useful when a request contains multiple updates and the client knows that the updates are independent, as might be the case when DSMLv2 is used to bulk-load a directory. It is also useful when a request contains multiple queries and no updates.

In a *BatchRequest* with *processing*=‶parallel″ and *responseOrder*=”unordered”, the client MUST specify a unique *requestID* for each individual request in the envelope. In this case, the server MAY return the responses in any order within the *BatchResponse* envelope – for example, in the order in which the operations complete, to improve server efficiency. If the client fails to specify a *requestID* for each request, the server MUST return an *errorResponse* with *type*=”malformedRequest”.

Resuming on error

A *BatchRequest* element MAY contain the optional XML-attribute *onError*, which influences how the server responds to failures while processing request elements. The valid values are: *exit* and *resume*. If this attribute is omitted, the default value is *exit*.

Example:

```
<batchRequest xmlns="urn:oasis:names:tc:DSML:2:0:core"
                 onError="resume">
 …
</batchRequest>
```

In a *BatchRequest* with *onError*=‶exit″, the server stops executing request elements as soon as one request element fails, and the response that is sent implicitly includes a *notAttempted* response for all requests that do not otherwise have a response.

If *processing*=‶parallel” and *onError*=‶exit”, the server stops initiating execution of *new* request elements as soon as one request element fails. Because of the parallelism, several executions might fail, and a request that the server did not attempt might precede a request that the server executed (with success or failure).

[5] A provider or server *could* detect that the document contains no updates and execute the queries in parallel without the client providing explicit directions, but is not required to do so.

When using positional correspondence between requests and responses, the provider/server MAY need to return responses for requests that it did not attempt. If the provider does not attempt to execute a request element, but needs to provide a response in order to maintain positional correspondence, it generates an *errorResponse* with *type*=“notAttempted”.

DSMLv2 Request with parallel execution containing a request that fails:

```
<batchRequest xmlns="urn:oasis:names:tc:DSML:2:0:core"
                processing="parallel" onError="resume">
  <modifyRequest>…</modifyRequest>
  <addRequest>…</addRequest>
  <delRequest>…</delRequest>
  <addRequest>…</addRequest>
</batchRequest>
```

DSMLv2 Response – two requests not successful

```
<batchResponse xmlns="urn:oasis:names:tc:DSML:2:0:core">
  <modifyResponse>…</modifyResponse>
  <errorResponse type="notAttempted"/>
  <delResponse>
    <resultCode code=”32” descr=”noSuchObject”/>
  </delResponse>
</batchResponse>
```

In a *BatchRequest* with *onError*=“resume*”*, the server executes the remaining request elements even though one or more requests have failed. This form of processing is most useful when *processing*=“parallel*”.*

Even when *processing*=“parallel*”*, the syntax checking of a request document is performed sequentially. The provider does not attempt any requests that follow the first syntax error in the document.

5. LDAP Operations

With the exception of *extendedRequest*, each individual request element contains:

- A *dn* attribute (as in DSMLv1) containing a distinguished name.
- Zero or more *control* elements representing LDAP Controls.

Here are few examples of LDAP request elements:

```
<batchRequest xmlns:xsd=”http://www.w3.org/2001/XMLSchema”
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns="urn:oasis:names:tc:DSML:2:0:core">
  <modifyRequest dn="CN=Joe Smith, OU=Dev, DC=Example, DC=Com">
    ...
  </modifyRequest>
  <addRequest  dn="OU=Sales,DC=Example, DC=Com">
    ...
  </addRequest>
  <delRequest dn="CN=Alice,OU=HR,DC=Example,DC=Com">
     <control>...</control>
     <control>...</control>
  </delRequest>
  <searchRequest>
```

```
    <control>...</control>
     ...
  </searchRequest>
</batchRequest>
```

Here is an example of an LDAP Control:[6]

```
<control type="1.2.840.113556.1.4.619" criticality="true">
  <controlValue xsi:type="xsd:base64Binary">RFNNTHYyLjAgcm9ja3MhIQ==</controlValue>
</control>
```

XML-Schema for *control* is here.

Here are few examples of LDAP response elements:

```
<batchResponse xmlns="urn:oasis:names:tc:DSML:2:0:core">
  ...
  <modifyResponse>
    <resultCode code=”53” descr="unwillingToPerform"/>
    <errorMessage>System Attribute may not be modified</errorMessage>
  </modifyResponse>

  <addResponse>
    <resultCode code=”0” descr="success”/>
  </addResponse>

  <addResponse>
    <control>...</control>
    <control>...</control>
    <resultCode code=”0” descr="success"/>
  </addResponse>

  ...
</batchResponse>
```

The *matchedDN* and *errorMessage* elements are optional and default to the empty string.

The *resultCode* element has an optional *descr* attribute. If the server supplies this attribute, its value is the RFC 2251 text representation of the result code (“success”, “operationsError”, etc.), supplied for debugging convenience.

Like LDAP Request elements, LDAP response elements MAY contain zero or more controls[7].

The remainder of this section describes the encoding of each LDAP operation. Refer to RFC 2251 for the semantics of LDAP operations.

[6] Note that as of this writing some popular XML parsers fail to properly validate this usage of xsi:type. However, it is explicitly allowed by XML Schema – see http://www.w3.org/TR/xmlschema-2/#union-datatypes.

[7] With the exception of *searchResponse*, for reasons that will become clear in Section 5.3 .

Values

The definition of *DsmlValue* permits the following types: UTF-8, base64Binary, and any URI. The URI type is used to indicate that the contents of the value are to be found at a location defined by the URI. For example, rather than sending a certificate as a base64Binary attribute value, a URI could be supplied that serves to locate the certificate on a certificate server. The ultimate value stored in the directory is the result of resolving the URI to establish that value. If it is intended to actually store a URI then it is appropriate to encode the URI with the UTF-8 value type.

Each binding MUST indicate how URIs are to be processed. For example, a binding MAY indicate that only 'http', 'ftp', 'file' or 'ldap' URIs are accepted and that they are processed only by the client provider. Another binding might specify that a DSMLv2 server for that binding will resolve the URI.

5.1 Auth

The *authRequest* provides a means for a client to indicate that access control for the following requests is to be interpreted as though the requests are performed by the security principal identified by the principal attribute. The value of the principal attribute is an *authzId*, as defined by [RFC 2829]. This can be useful if the DSMLv2 server (or an LDAP server to which the DSMLv2 server connects) is capable of supporting proxy authorization [ID-ProxyAuth].

At most one *authRequest* MAY occur within a *BatchRequest* and if it does occur, it MUST be the first request. If *authRequest* operations are not supported by the server to which the *BatchRequest* is sent, then the server MUST NOT process the following requests and MUST return a *BatchResponse* with an *authResponse* containing an *LDAPResultCode* of 'authMethodNotSupported'. If *authRequest* operations are supported, then if there are access rights errors, processing proceeds as for a *BatchRequest* without an *authRequest* – i.e., an appropriate *errorResponse* is generated, etc.

Example of *authRequest*:

```
<authRequest principal="dn:CN=Bob Rush,OU=Dev,DC=Example,DC=COM"/>
```

XML-Schema for *authRequest* is here.

Example of *authResponse*:

```
<authResponse>
  <resultCode code="0"/>
</authResponse>
```

authResponse is an *LDAPResult*.

5.2 Modify

DSMLv2 specifies each attribute modification by attaching an *operation* attribute to a *modification* element. As in LDAP, an *operation* can be *add*, *delete*, or *replace*.

Example of *modifyRequest*:

```
<modifyRequest dn="CN=Bob Rush,OU=Dev,DC=Example,DC=COM">
  <modification name="telephoneNumber" operation="replace">
    <value>536 354 2343</value>
    <value>234 212 4534</value>
  </modification>
  <modification name="sn" operation="replace">
    <value>Rush</value>
  </modification>
  <modification name="directReport" operation="add">
    <value>CN=John Smith, DC=microsoft, DC=com</value>
  </modification>
</modifyRequest>
```

XML-Schema for *modifyRequest* is here.

Example of *modifyResponse*:

```
<modifyResponse>
  <resultCode code="53" descr="unwillingToPerform"/>
  <errorMessage>System Attribute may not be modified</errorMessage>
</modifyResponse>
```

modifyResponse is an *LDAPResult*.

5.3 Search

The DSMLv2 search encoding is based on the LDAP search encoding, but with some changes as described in Section 2. In the *searchRequest* encoding:

- *baseObject*. Following DSMLv1 conventions, the distinguished name of the search base is expressed as the XML attribute *dn*.

  Example:
  <searchRequest dn=”OU=Marketing,DC=Example,DC=COM” />

- *sizeLimit, timeLimit, typesOnly*. These elements default to 0, 0, and FALSE respectively.

- *attributes*. In RFC 2251, *attributes* is a sequence of attribute names, which is translated into a sequence of elements containing attribute names.
  Example:
  ```
  <attributes>
    <attribute name=”sn”/ >
    <attribute name=”givenName”/>
    <attribute name=”title”/>
  </attributes>
  ```

SearchRequest example:

```
<searchRequest dn="ou=Marketing,dc=microsoft,dc=com"
```

```
                  scope="singleLevel"
                  derefAliases="neverDerefAliases"
                  sizeLimit="1000">
  <control type="1.2.840.113556.1.4.612" criticality="true">
    <controlValue xsi:type="xsd:base64Binary">
      U2VhcmNoIFJlcXVlc3QgRXhhbXBsZQ==
    </controlValue>
  </control>
  <control type="1.2.840.113556.1.4.643" criticality="true">
    <controlValue xsi:type="xsd:base64Binary">
      TWljcm9zb2Z0IEFjdGl2ZSBEaXJlY3Rvcnk=
    </controlValue>
  </control>
  <filter><substrings  name="sn"><final>john</final></substrings></filter>
</searchRequest>
```

XML-Schema for *searchRequest* is here.

The response to a *searchRequest* is logically called a *searchResponse*. According to RFC 2251, a search response contains a) zero to many *searchResultEntry*, b) zero to many *searchResultReference* and c) one *searchResultDone*. DSMLv2 permits wrapping all of these related elements into one *searchResponse* envelope.

Each *searchResultEntry*, *searchResultReference*, and *searchResultDone* MAY have zero or more LDAP controls, consistent with RFC 2251.

searchResultEntry (with terminating searchResultDone) example:

```
<searchResponse>
  <searchResultEntry dn="OU=Development,DC=Example,DC=COM">
    <attr name="allowedAttributeEffective">
      <value>description</value>
      <value>ntSecurityDescriptor</value>
      <value>wwwHomepage</value>
    </attr>
  </searchResultEntry>
  <searchResultEntry dn="CN=David,OU=HR,DC=Example,DC=COM">
    <attr name="objectclass"><value>person</value></attr>
    <attr name="sn"><value>Johnson</value></attr>
    <attr name="givenName"><value>David</value></attr>
    <attr name="title"><value>Program Manager</value></attr>
  </searchResultEntry>
  <searchResultEntry dn="CN=JSmith, OU=Finance,DC=Example,DC=COM">
    <attr name="objectclass"><value>top</value></attr>
    <attr name="objectclass"><value>person</value></attr>
    <attr name="objectclass"><value>organizationalPerson</value></attr>
    <attr name="sn"><value>Smith</value></attr>
  </searchResultEntry>
  <searchResultDone>
    <control type="1.2.840.113556.1.4.621" criticality="false">
      <controlValue xsi:type="xsd:base64Binary">
        U2VhcmNoIFJlcXVlc3QgRXhhbXBsZQ==
      </controlValue>
    </control>
    <resultCode code="0"/>
  </searchResultDone>
</searchResponse>
```

XML-Schema for *searchResultEntry* is here.

searchResultReference example:

```
<searchResponse>
  <searchResultReference>
    <ref>ldap://srv01.example.com/OU=Marketing,DC=Example,DC=COM</ref>
    <ref>ldap://srv05.fabrikam.com/DC=Fabrikam,DC=COM</ref>
  </searchResultReference>
      …
</searchResponse>
```

XML-Schema for *searchResultReference* is here.

searchResultDone is illustrated above in the *searchResultEntry* example. *searchResultDone* is an *LDAPResult*.

Note that it MAY be useful to limit the size of the *searchResponse* via *sizeLimit* or the use of a virtual list view control [ID-VLV]. Further, this specification does not introduce any limitation regarding the streaming processing of requests or responses including particularly *searchResponse*. In other words it is not necessary that an implementation build the entire *searchResponse* prior to sending the initial segment of that response.

5.4 Add

A DSMLv2 *addRequest* uses an encoding style analogous to that of the *searchResultEntry* to describe the object to be added.

Example of addRequest:

```
<addRequest dn="CN=Alice,OU=HR,DC=Example,DC=COM">
  <attr name="objectclass"><value>top</value></attr>
  <attr name="objectclass"><value>person</value></attr>
  <attr name="objectclass"><value>organizationalPerson</value></attr>
  <attr name="sn"><value>Johnson</value></attr>
  <attr name="givenName"><value>Alice</value></attr>
  <attr name="title"><value>Software Design Engineer</value></attr>
</addRequest>
```

XML-Schema for *addRequest* is here.

Example of *addResponse*:

```
<addResponse>
  <resultCode code="0"/>
  <errorMessage>completed</errorMessage>
</addResponse>
```

addResponse is an *LDAPResult*.

5.5 Delete

Examples of *delRequest*:

```
<delRequest dn="CN=Bob,OU=HR,DC=Example,DC=COM" />

<delRequest dn="OU=HR,DC=Example,DC=COM" >
   <control type="1.2.840.113556.1.4.805"/>
</delRequest>
```

XML-Schema for *delRequest* is here.

Example of *delResponse*:

```
<delResponse matchedDN="OU=HR,DC=Example,DC=COM">
   <resultCode code="32" descr="noSuchObject"/>
   <errorMessage>DSDEL::230234</errorMessage>
</delResponse>
```

delResponse is an LDAPResult.

5.6 ModifyDN

Example of *modDNRequest*.

```
<modDNRequest dn="CN=Alice Johnson,DC=Example,DC=COM"
                          newrdn="CN=Alice Weiss"
                          deleteoldrdn="true"
                          newSuperior="OU=Marketing,DC=Example,DC=COM"/>
```

XML-Schema for *modDNRequest* is here.

Example of modDNResponse.

```
<modDNResponse>
   <resultCode code="0" descr="success"/>
</modDNResponse>
```

modDNResponse is an *LDAPResult*.

5.7 Compare

Example of *compareRequest*:

```
<compareRequest dn="CN=Johnson,OU=HR, DC=Example,DC=COM">
   <assertion name="sn"><value>Johnson</value></assertion>
</compareRequest>
```

XML-Schema for *compareRequest* is here.

Example of *compareResponse*:

```
<compareResponse>
```

```
  <resultCode code="6" descr="compareTrue"/>
</compareResponse>
```

The *compareResponse* is an *LDAPResult*.

5.8 Extended Operation

Example of *extendedRequest*:

```
<extendedRequest>
  <requestName>1.3.563.52.424</requestName>
  <requestValue xsi:type="xsd:base64Binary">TFNNTHYyLjAgcm9ja3MhIQ==</requestValue>
</extendedRequest>
```

XML-Schema for *extendedRequest* is here.

Example of *extendedResponse*:

```
<extendedResponse>
  <resultCode code="0"/>
  <response xsi:type="xsd:base64Binary">RFNNTHYyLjAgcm9ja3MhIQ==</response>
</extendedResponse>
```

XML-Schema for *extendedResponse* is here.

6. SOAP Request/Response Binding

The following describes the DSMLv2 SOAP [W3C SOAP] request/response binding, using HTTP/1.1 and HTTPS/1.1 as a transport for DSMLv2 requests and responses. The version of SOAP for this binding is SOAP 1.1. The use of other transports for SOAP, e.g., SMTP, would be subject to additional considerations not addressed here. Further, other DSMLv2 bindings that incorporate SOAP are permitted by this specification.

The namespace for DSMLv2 is "urn:oasis:names:tc:DSML:2:0:core". This namespace is used at the top-level element of the <body> of each SOAP request and response. Default namespace designations MAY be used.

All SOAP requests and responses in this binding MUST use the xml encoding "UTF-8".

Each SOAP request body contains a single *batchRequest*. A SOAP node SHOULD indicate in the 'SOAPAction' header field the element name of the top-level element in the <body> of the SOAP request.

Each SOAP response body contains a single *batchResponse*.

A SOAP Fault is used only when an error occurs outside the scope of DSMLv2 processing. For example, the SOAP Server is not able to find or connect to a DSMLv2 server to process a DSMLv2 document. If errors happen during DSMLv2 processing, then they are conveyed as a DSMLv2 response document in the SOAP response message.

For example:

```
<!--  **** DSMLv2 Request ****** -->
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/">
  <se:Body xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core">
    <dsml:batchRequest>
      <dsml:modifyRequest>…</dsml:modifyRequest>
      <dsml:addRequest>…</dsml:addRequest>
       …
    </dsml:batchRequest>
  </se:Body>
</se:Envelope>


<!--  **** DSMLv2 Response ****** -->
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/">
  <se:Body xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core">
    <dsml:batchResponse>
      <dsml:modifyResponse> …</dsml:modifyResponse>
      <dsml:addResponse>…</dsml:modifyResponse>
       …
    </dsml:batchResponse>
  </se:Body>
</se:Envelope>


<!--  **** SOAP Fault ****** -->
<se:Envelope xmlns:se=http://schemas.xmlsoap.org/soap/envelope/>
  <se:Body>
    <se:Fault>
      <faultcode>se:Server</faultcode>
      <faultstring>Server Error</faultstring>
      <detail>Cannot connect to a DSMLv2 server</detail>
    </se:Fault>
  </se:Body>
</se:Envelope>
```

This binding does not specify any SOAP headers.

Authentication in this binding is dependent upon the transport that is used with SOAP. At present, the SOAP specification does not define authentication mechanisms. A few SOAP security proposals have been submitted for consideration. If in the future a SOAP security standard emerges, the DSMLv2 working group will revisit to determine whether DSMLv2 should adopt the standard. Today, in the absence of a SOAP security standard, the DSMLv2 working group sets a guideline for SOAP authentication over HTTP. When using HTTP 1.1 the authentication information is transferred in accordance with [RFC 2617] which covers Basic and Digest forms. It is recommended that TLS 1.0 [RFC 2246] be used in particular with the Basic form of authentication.

A minimal implementation supports *DsmlValue* URIs of type file, which are evaluated by the client provider using the security context associated with the client. Individual implementations MAY support additional URI types. If the client provider is unable to resolve the URI to a value that can transferred to the server, then the provider MUST return an *errorResponse* with *type*="unresolvableURI".

7. File Binding

The file binding is an alternative to the LDAP Data Interchange Format (LDIF) described by [RFC 2849]. Its primary advantages over LDIF are:

- Use of XML, which is more natural for many clients to generate and to parse than LDIF. Also benefits from the comparative wealth of tools.
- Formalization of output on error conditions, such as in the event the directory server is unavailable or the directory server returns an LDAP error.

With the file binding, the DSMLv2 "server" is a command-line program, as is typical for LDIF. The client that invokes the "server" program runs on the same computer as the server, and the input and output of the server are files consisting of DSMLv2 documents. The DSMLv2 server uses LDAP, another DSMLv2 transport, or some other mechanism to communicate with the directory service, which may be running on the same computer or on a different computer.

The top-level document for the input file is an element of type *BatchRequest* with name *batchRequest*. The top-level document for the output file is an element of type *BatchResponse* with name *batchResponse*.

A minimal implementation supports *DsmlValue* URIs of type file, which are evaluated by the command-line program using the security context associated with the process running the command-line program. Individual implementations MAY support additional URI types. If the client provider is unable to resolve the URI to a value that can transferred to the server, then the command-line program MUST return an *errorResponse* with *type*=”unresolvableURI”.

By default the file binding authenticates to the directory using the user identity with which the command-line program was invoked. Individual implementations MAY support or require specification of explicit credentials on the command line.

8. Extensibility Guidelines

This specification includes two bindings: 6. SOAP Request/Response Binding; and 7. File Binding. These two bindings do not exhaust the list of reasonable bindings for DSMLv2. There are many other applications of DSMLv2 that will use other transports such as BEEP [RFC 3080] or a message bus such as MQ-series [MQS] or iMQ [iMQ]. This section identifies some of the issues that MUST be considered when developing a new binding of DSMLv2.

A binding MUST specify what DSMLv2 top-level elements it will incorporate. For example, a binding might use the *DsmlRequests* and *DsmlResponses* with an envelope definition suited to a particular application of a particular transport. Alternatively a binding might specify that it uses the *BatchRequest* and *BatchResponse* envelopes.

A binding MUST specify the method by which a URI type *DsmlValue* is resolved and which forms of URI are supported by the binding. For example, the 6. SOAP

Request/Response Binding specifies that the URI type values are resolved by the client provider and that the permitted forms are limited to 'file'.

A binding MUST identify how authentication is performed so that a security principal MAY be associated with the various requests that will be issued.

A binding MAY identify a 'session' mechanism that will permit independent sequences of requests to be associated with authentication information or that would permit sets of responses, for example from a persistent search, to be associated with a particular request.

9. Bibliography

[ID-ProxyAuth] Rob Weltman; "Proxy Authorization Control."
http://search.ietf.org/internet-drafts/draft-weltman-ldapv3-proxy-07.txt

[ID-VLV] “LDAP Extensions for Scrolling View Browsing of Search Results.”
http://search.ietf.org/internet-drafts/draft-ietf-ldapext-ldapv3-vlv-05.txt

[iMQ] "iPlanet Message Queue for Java."
http://docs.iplanet.com/docs/manuals/javamq.html

[MQS] IBM MQ-Series

[RFC 2119] "Key words for use in RFCs to Indicate Requirement Levels."
http://www.ietf.org/rfc/rfc2119.txt

[RFC 2246] "The TLS Protocol"
http://www.ietf.org/rfc/rfc2246.txt

[RFC 2617] "HTTP Authentication: Basic and Digest Access Authentication"
http://www.ietf.org/rfc/rfc2617.txt

[RFC 2829] "Authentication Methods for LDAP"
http://www.ietf.org/rfc/rfc2829.txt

[RFC 2849] "The LDAP Data Interchange Format (LDIF)"
http://www.ietf.org/rfc/rfc2849.txt

[RFC 3080] "The Blocks Extensible Exchange Protocol Core."
http://www.ietf.org/rfc/rfc3080.txt

[W3C SOAP] "SOAP Version 1.2 Part 1: Messaging Framework Working Draft"
http://www.w3c.org/TR2001/WD-soap12-part1-20011002

10. Appendix

DSMLv2 Schema

```
<xsd:schema targetNamespace="urn:oasis:names:tc:DSML:2:0:core"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="urn:oasis:names:tc:DSML:2:0:core"
elementFormDefault="qualified">
    <!-- DSML Requests -->
    <xsd:group name="DSMLRequests">
        <xsd:choice>
            <xsd:element name="authRequest" type="AuthRequest"/>
            <xsd:group ref="BatchRequests"/>
        </xsd:choice>
    </xsd:group>
    <xsd:group name="BatchRequests">
        <xsd:choice>
            <xsd:element name="searchRequest" type="SearchRequest"/>
            <xsd:element name="modifyRequest" type="ModifyRequest"/>
            <xsd:element name="addRequest" type="AddRequest"/>
            <xsd:element name="delRequest" type="DelRequest"/>
            <xsd:element name="modDNRequest" type="ModifyDNRequest"/>
            <xsd:element name="compareRequest" type="CompareRequest"/>
            <xsd:element name="abandonRequest" type="AbandonRequest"/>
            <xsd:element name="extendedRequest" type="ExtendedRequest"/>
        </xsd:choice>
    </xsd:group>
    <!-- DSML Responses -->
    <xsd:group name="DSMLResponses">
        <xsd:choice>
            <xsd:element name="authResponse" type="LDAPResult"/>
            <xsd:element name="searchResultEntry" type="SearchResultEntry"/>
            <xsd:element name="searchResultReference" type="SearchResultReference"/>
            <xsd:element name="searchResultDone" type="LDAPResult"/>
            <xsd:element name="modifyResponse" type="LDAPResult"/>
            <xsd:element name="addResponse" type="LDAPResult"/>
            <xsd:element name="delResponse" type="LDAPResult"/>
            <xsd:element name="modDNResponse" type="LDAPResult"/>
            <xsd:element name="compareResponse" type="LDAPResult"/>
            <xsd:element name="extendedResponse" type="ExtendedResponse"/>
            <xsd:element name="errorResponse" type="ErrorResponse"/>
        </xsd:choice>
    </xsd:group>
    <!--  *************** Batch Envelopes ********************* -->
    <xsd:element name="batchRequest" type="BatchRequest"/>
    <xsd:element name="batchResponse" type="BatchResponse"/>
    <!-- **** Batch Request Envelope **** -->
    <xsd:complexType name="BatchRequest">
        <xsd:sequence>
            <xsd:element name="authRequest" type="AuthRequest" minOccurs="0"/>
            <xsd:group ref="BatchRequests" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="requestID" type="RequestID" use="optional"/>
        <xsd:attribute name="processing" use="optional" default="sequential">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="sequential"/>
                    <xsd:enumeration value="parallel"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="responseOrder" use="optional" default="sequential">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="sequential"/>
                    <xsd:enumeration value="unordered"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="onError" use="optional" default="exit">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="resume"/>
                    <xsd:enumeration value="exit"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
    <!-- **** Batch Response Envelope **** -->
    <xsd:complexType name="BatchResponse">
        <xsd:sequence>
            <xsd:group ref="BatchResponses" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="requestID" type="RequestID" use="optional"/>
    </xsd:complexType>
    <!-- **** Batch Responses **** -->
    <xsd:group name="BatchResponses">
        <xsd:choice>
            <xsd:element name="searchResponse" type="SearchResponse"/>
            <xsd:element name="authResponse" type="LDAPResult"/>
            <xsd:element name="modifyResponse" type="LDAPResult"/>
            <xsd:element name="addResponse" type="LDAPResult"/>
            <xsd:element name="delResponse" type="LDAPResult"/>
            <xsd:element name="modDNResponse" type="LDAPResult"/>
            <xsd:element name="compareResponse" type="LDAPResult"/>
            <xsd:element name="extendedResponse" type="ExtendedResponse"/>
            <xsd:element name="errorResponse" type="ErrorResponse"/>
        </xsd:choice>
    </xsd:group>
    <!-- **** Search Response **** -->
    <xsd:complexType name="SearchResponse">
        <xsd:sequence>
            <xsd:element name="searchResultEntry" type="SearchResultEntry" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element name="searchResultReference" type="SearchResultReference" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element name="searchResultDone" type="LDAPResult"/>
        </xsd:sequence>
        <xsd:attribute name="requestID" type="RequestID" use="optional"/>
    </xsd:complexType>
    <!-- ***** DsmlDN ***** -->
    <xsd:simpleType name="DsmlDN">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <!-- ***** DsmlRDN ***** -->
    <xsd:simpleType name="DsmlRDN">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <!-- ***** Request ID ***** -->
    <xsd:simpleType name="RequestID">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <!-- ***** AttributeDescriptionValue ***** -->
    <xsd:simpleType name="AttributeDescriptionValue">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="((([0-2](\.[0-9]+)+)|([a-zA-Z]+([a-zA-Z0-9]|[-])*))(;([a-zA-Z0-9]|[-])+)*)"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="NumericOID">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="[0-2]\.[0-9]+(\.[0-9]+)*"/>
        </xsd:restriction>
    </xsd:simpleType>
    <!-- ***** MAX Integer ***** -->
```

```
<xsd:simpleType name="MAXINT">
    <xsd:restriction base="xsd:unsignedInt">
        <xsd:maxInclusive value="2147483647"/>
    </xsd:restriction>
</xsd:simpleType>
<!-- **** DSML Value **** -->
<xsd:simpleType name="DsmlValue">
    <xsd:union memberTypes="xsd:string xsd:base64Binary xsd:anyURI"/>
</xsd:simpleType>
<!-- **** DSML Control **** -->
<xsd:complexType name="Control">
    <xsd:sequence>
        <xsd:element name="controlValue" type="xsd:anyType" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="type" type="NumericOID" use="required"/>
    <xsd:attribute name="criticality" type="xsd:boolean" use="optional" default="false"/>
</xsd:complexType>
<!-- **** DSML Filter **** -->
<xsd:complexType name="Filter">
    <xsd:group ref="FilterGroup"/>
</xsd:complexType>
<xsd:group name="FilterGroup">
    <xsd:sequence>
        <xsd:choice>
            <xsd:element name="and" type="FilterSet"/>
            <xsd:element name="or" type="FilterSet"/>
            <xsd:element name="not" type="Filter"/>
            <xsd:element name="equalityMatch" type="AttributeValueAssertion"/>
            <xsd:element name="substrings" type="SubstringFilter"/>
            <xsd:element name="greaterOrEqual" type="AttributeValueAssertion"/>
            <xsd:element name="lessOrEqual" type="AttributeValueAssertion"/>
            <xsd:element name="present" type="AttributeDescription"/>
            <xsd:element name="approxMatch" type="AttributeValueAssertion"/>
            <xsd:element name="extensibleMatch" type="MatchingRuleAssertion"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:group>
<xsd:complexType name="FilterSet">
    <xsd:sequence>
        <xsd:group ref="FilterGroup" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AttributeValueAssertion">
    <xsd:sequence>
        <xsd:element name="value" type="DsmlValue"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="AttributeDescriptionValue" use="required"/>
</xsd:complexType>
<xsd:complexType name="AttributeDescription">
    <xsd:attribute name="name" type="AttributeDescriptionValue" use="required"/>
</xsd:complexType>
<xsd:complexType name="SubstringFilter">
    <xsd:sequence>
        <xsd:element name="initial" type="DsmlValue" minOccurs="0"/>
        <xsd:element name="any" type="DsmlValue" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="final" type="DsmlValue" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="AttributeDescriptionValue" use="required"/>
</xsd:complexType>
<xsd:complexType name="MatchingRuleAssertion">
    <xsd:sequence>
        <xsd:element name="value" type="DsmlValue"/>
    </xsd:sequence>
    <xsd:attribute name="dnAttributes" type="xsd:boolean" use="optional" default="false"/>
    <xsd:attribute name="matchingRule" type="xsd:string" use="optional"/>
    <xsd:attribute name="name" type="AttributeDescriptionValue" use="optional"/>
```

```
</xsd:complexType>
<!--  *************** DSML MESSAGE ******************** -->
<xsd:complexType name="DsmlMessage">
    <xsd:sequence>
        <xsd:element name="control" type="Control" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="requestID" type="RequestID" use="optional"/>
</xsd:complexType>
<!--  *************** LDAP RESULT ********************* -->
<xsd:simpleType name="LDAPResultCode">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="success"/>
        <xsd:enumeration value="operationsError"/>
        <xsd:enumeration value="protocolError"/>
        <xsd:enumeration value="timeLimitExceeded"/>
        <xsd:enumeration value="sizeLimitExceeded"/>
        <xsd:enumeration value="compareFalse"/>
        <xsd:enumeration value="compareTrue"/>
        <xsd:enumeration value="authMethodNotSupported"/>
        <xsd:enumeration value="strongAuthRequired"/>
        <xsd:enumeration value="referral"/>
        <xsd:enumeration value="adminLimitExceeded"/>
        <xsd:enumeration value="unavailableCriticalExtension"/>
        <xsd:enumeration value="confidentialityRequired"/>
        <xsd:enumeration value="saslBindInProgress"/>
        <xsd:enumeration value="noSuchAttribute"/>
        <xsd:enumeration value="undefinedAttributeType"/>
        <xsd:enumeration value="inappropriateMatching"/>
        <xsd:enumeration value="constraintViolation"/>
        <xsd:enumeration value="attributeOrValueExists"/>
        <xsd:enumeration value="invalidAttributeSyntax"/>
        <xsd:enumeration value="noSuchObject"/>
        <xsd:enumeration value="aliasProblem"/>
        <xsd:enumeration value="invalidDNSyntax"/>
        <xsd:enumeration value="aliasDerefencingProblem"/>
        <xsd:enumeration value="inappropriateAuthentication"/>
        <xsd:enumeration value="invalidCredentials"/>
        <xsd:enumeration value="insufficientAccessRights"/>
        <xsd:enumeration value="busy"/>
        <xsd:enumeration value="unavailable"/>
        <xsd:enumeration value="unwillingToPerform"/>
        <xsd:enumeration value="loopDetect"/>
        <xsd:enumeration value="namingViolation"/>
        <xsd:enumeration value="objectClassViolation"/>
        <xsd:enumeration value="notAllowedOnNonLeaf"/>
        <xsd:enumeration value="notAllowedOnRDN"/>
        <xsd:enumeration value="entryAlreadyExists"/>
        <xsd:enumeration value="objectClassModsProhibited"/>
        <xsd:enumeration value="affectMultipleDSAs"/>
        <xsd:enumeration value="other"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="ResultCode">
    <xsd:attribute name="code" type="xsd:int" use="required"/>
    <xsd:attribute name="descr" type="LDAPResultCode" use="optional"/>
</xsd:complexType>
<xsd:complexType name="LDAPResult">
    <xsd:complexContent>
        <xsd:extension base="DsmlMessage">
            <xsd:sequence>
                <xsd:element name="resultCode" type="ResultCode"/>
                <xsd:element name="errorMessage" type="xsd:string" minOccurs="0"/>
                <xsd:element name="referral" type="xsd:anyURI" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="matchedDN" type="DsmlDN" use="optional"/>
        </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ErrorResponse">
        <xsd:sequence>
            <xsd:element name="message" type="xsd:string" minOccurs="0"/>
            <xsd:element name="detail" minOccurs="0">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:any/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="requestID" type="RequestID" use="optional"/>
        <xsd:attribute name="type">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="notAttempted"/>
                    <xsd:enumeration value="couldNotConnect"/>
                    <xsd:enumeration value="connectionClosed"/>
                    <xsd:enumeration value="malformedRequest"/>
                    <xsd:enumeration value="gatewayInternalError"/>
                    <xsd:enumeration value="authenticationFailed"/>
                    <xsd:enumeration value="unresolvableURI"/>
                    <xsd:enumeration value="other"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
    <!-- *************** Auth ********************* -->
    <xsd:complexType name="AuthRequest">
        <xsd:complexContent>
            <xsd:extension base="DsmlMessage">
                <xsd:attribute name="principal" type="xsd:string" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- *************** Search ********************* -->
    <xsd:complexType name="AttributeDescriptions">
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="attribute" type="AttributeDescription"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="SearchRequest">
        <xsd:complexContent>
            <xsd:extension base="DsmlMessage">
                <xsd:sequence>
                    <xsd:element name="filter" type="Filter"/>
                    <xsd:element name="attributes" type="AttributeDescriptions" minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="dn" type="DsmlDN" use="required"/>
                <xsd:attribute name="scope" use="required">
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                            <xsd:enumeration value="baseObject"/>
                            <xsd:enumeration value="singleLevel"/>
                            <xsd:enumeration value="wholeSubtree"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="derefAliases" use="required">
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                            <xsd:enumeration value="neverDerefAliases"/>
                            <xsd:enumeration value="derefInSearching"/>
                            <xsd:enumeration value="derefFindingBaseObj"/>
                    <xsd:enumeration value="derefAlways"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="sizeLimit" type="MAXINT" use="optional" default="0"/>
        <xsd:attribute name="timeLimit" type="MAXINT" use="optional" default="0"/>
        <xsd:attribute name="typesOnly" type="xsd:boolean" use="optional" default="false"/>
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<!-- ***** Search Result Entry ***** -->
<xsd:complexType name="SearchResultEntry">
    <xsd:complexContent>
        <xsd:extension base="DsmlMessage">
            <xsd:sequence>
                <xsd:element name="attr" type="DsmlAttr" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="dn" type="DsmlDN" use="required"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="DsmlAttr">
    <xsd:sequence>
        <xsd:element name="value" type="DsmlValue" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="AttributeDescriptionValue" use="required"/>
</xsd:complexType>
<xsd:complexType name="DsmlModification">
    <xsd:sequence>
        <xsd:element name="value" type="DsmlValue" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="AttributeDescriptionValue" use="required"/>
    <xsd:attribute name="operation" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="add"/>
                <xsd:enumeration value="delete"/>
                <xsd:enumeration value="replace"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>
<!-- ***** Search Result Reference ***** -->
<xsd:complexType name="SearchResultReference">
    <xsd:complexContent>
        <xsd:extension base="DsmlMessage">
            <xsd:sequence>
                <xsd:element name="ref" type="xsd:anyURI" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<!-- ************* MODIFY ******************** -->
<xsd:complexType name="ModifyRequest">
    <xsd:complexContent>
        <xsd:extension base="DsmlMessage">
            <xsd:sequence>
                <xsd:element name="modification" type="DsmlModification" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="dn" type="DsmlDN" use="required"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<!--  *************** ADD ********************* -->
<xsd:complexType name="AddRequest">
```

```
        <xsd:complexContent>
            <xsd:extension base="DsmlMessage">
                <xsd:sequence>
                    <xsd:element name="attr" type="DsmlAttr" minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="dn" type="DsmlDN" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- *************** DELETE ********************* -->
    <xsd:complexType name="DelRequest">
        <xsd:complexContent>
            <xsd:extension base="DsmlMessage">
                <xsd:attribute name="dn" type="DsmlDN" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- *************** MODIFY DN ********************* -->
    <xsd:complexType name="ModifyDNRequest">
        <xsd:complexContent>
            <xsd:extension base="DsmlMessage">
                <xsd:attribute name="dn" type="DsmlDN" use="required"/>
                <xsd:attribute name="newrdn" type="DsmlRDN" use="required"/>
                <xsd:attribute name="deleteoldrdn" type="xsd:boolean" use="optional" default="true"/>
                <xsd:attribute name="newSuperior" type="DsmlDN" use="optional"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- ************* COMPARE ******************** -->
    <xsd:complexType name="CompareRequest">
        <xsd:complexContent>
            <xsd:extension base="DsmlMessage">
                <xsd:sequence>
                    <xsd:element name="assertion" type="AttributeValueAssertion"/>
                </xsd:sequence>
                <xsd:attribute name="dn" type="DsmlDN" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- ***** ABANDON ***** -->
    <xsd:complexType name="AbandonRequest">
        <xsd:complexContent>
            <xsd:extension base="DsmlMessage">
                <xsd:attribute name="abandonID" type="RequestID" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- ************* EXTENDED OPERATION ******************** -->
    <xsd:complexType name="ExtendedRequest">
        <xsd:complexContent>
            <xsd:extension base="DsmlMessage">
                <xsd:sequence>
                    <xsd:element name="requestName" type="NumericOID"/>
                    <xsd:element name="requestValue" type="xsd:anyType" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ExtendedResponse">
        <xsd:complexContent>
            <xsd:extension base="LDAPResult">
                <xsd:sequence>
                    <xsd:element name="responseName" type="NumericOID" minOccurs="0"/>
                    <xsd:element name="response" type="xsd:anyType" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- ********************END base SCHEMA ******************** -->
</xsd:schema>
```

11. OASIS legal statements

Directory Services Markup Language v2.0 Errata

This document contains clarifications and corrections to the DSMLv2 specification dated December 18, 2001, found at http://www.oasis-open.org/committees/dsml/docs/DSMLv2.doc.

Errata for March 25, 2002

4. Top-Level Structure

Syntax errors

The second sentence should read:

> In this case the DSMLv2 server MUST produce a response document to aid in debugging the client.

The third paragraph should read:

> If the server performs one or more directory operations on behalf of the client before detecting the syntax error, the server's response MUST contain the response element for each operation that it performed, followed by an *errorResponse* element. The server MUST NOT process requests containing or following a syntax error. For instance,

Failures

The footnote reading, "Obviously neither this nor the previous condition…" should be disregarded.

Parallel processing

The last sentence of this section should read:

> If the client fails to specify a *requestID* for a given request in a *batchRequest* in which *processing*="parallel", the server MUST proceed as if the request contains a syntax error.

Resuming on error

The last two sentences of the third paragraph should read:

> When using positional correspondence between requests and responses, the server MAY need to return responses for requests that it did not attempt. If the server does not attempt to execute a request element, but needs to provide a response in order to maintain positional correspondence, it MUST generate an *errorResponse* with *type*="notAttempted".

In the example for "*DSMLv2 Request with parallel execution containing a request that fails,*" the *batchRequest* should contain *onError*="exit", not *onError*="resume". The response caption should be "*DSMLv2 Response – two requests not attempted*".

The last paragraph should read:

Recall that the server does not attempt any requests that follow the first syntax error in the document. This applies even to *batchRequest* elements in which *processing*=”parallel” and *onError*=”resume”.

5. LDAP Operations

5.1 Auth

The last sentence of the second paragraph should read:

> If *authRequest* operations are supported, then if there are access rights errors, processing proceeds as for a *BatchRequest* containing a failure without an *authRequest* – i.e., an appropriate *errorResponse* is generated, etc.

If the server finds an *authRequest* element in a *BatchRequest* in which *processing*=”parallel”, the server MUST fully process the *authRequest* before beginning to process the remainder of the requests in the *BatchRequest*. In other words, the server MUST apply the effects of the *authRequest* to all other requests within the *BatchRequest*.