INTERNET-DRAFT
draft-reddy-dasl-protocol-04.txt

Expires May 18, 1999

November 18, 1998

Saveen Reddy, Microsoft
Dale Lowry, Novell
Surendra Reddy, Oracle
Rick Henderson, Netscape
Jim Davis, Xerox
Alan Babich, Filenet

DAV Searching & Locating

Status of this Memo

Distribution of this document is unlimited. Please send comments to the mailing list at <www-webdav-dasl@w3.org>, which may be joined by sending a message with subject "subscribe" to <www-webdav-dasl-request@w3.org>.

Discussions of the list are archived at <URL:http://www.w3.org/pub/WWW/Archives/Public/www-webdav-dasl>.

Abstract

This document specifies a set of methods, headers, and content-types composing DASL, an application of the HTTP/1.1 protocol to efficiently search for DAV resources based upon a set of client-supplied criteria.

Table of Contents

1. INTRODUCTION

1.1. DASL

This document defines DAV Searching & Locating (DASL), an application of HTTP/1.1 forming a lightweight search protocol to transport queries and result sets and allows clients to make use of server-side search facilities. [DASLREQ] describes the motivation for DASL.

DASL will minimize the complexity of clients so as to facilitate widespread deployment of applications capable of utilizing the DASL search mechanisms.

DASL consists of:

- the SEARCH method,

- the DASL response header,

- the DAV:searchrequest XML element,

- the DAV:queryschema property,

- the DAV:basicsearch XML element and query grammar, and

- the DAV:basicsearchschema XML element.

1.2. Relationship to DAV

DASL relies on the resource and property model defined by [WebDAV]. DASL does not alter this model. Instead, DASL allows clients to access DAV-modeled resources through server-side search.

1.3. Terms

This draft uses the terms defined in [RFC2068], [WebDAV], and [DASLREQ].

1.4. Notational Conventions

The augmented BNF used by this document to describe protocol elements is exactly the same as the one described in Section 2.1 of [RFC2068]. Because this augmented BNF uses the basic production rules provided in Section 2.2 of [RFC2068], those rules apply to this document as well.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.5. An Overview of DASL at Work

One can express the basic usage of DASL in the following steps:

- The client constructs a query using the DAV:basicsearch grammar.

- The client invokes the SEARCH method on a resource that will perform the search (the search arbiter) and includes a text/xml request entity that contains the query.

- The search arbiter performs the query.

- The search arbiter sends the results of the query back to the client in the response. The server MUST send a text/xml entity that matches the [WebDAV] PROPFIND response.

2. THE SEARCH METHOD

2.1. Overview

The client invokes the SEARCH method to initiate a server-side search. The body of the request defines the query. The server MUST emit text/xml entity matching the [WebDAV] PROPFIND response.

The SEARCH method plays the role of transport mechanism for the query and the result set. It does not define the semantics of the query. The type of the query defines the semantics.

2.2. The Request

The client invokes the SEARCH method on the resource named by the Request-URI.

2.2.1. The Request-URI

The Request-URI identifies the search arbiter.

The SEARCH method per se defines no relationship between the arbiter and the scope of the search, rather the particular query grammar used in the query defines the relationship. For example, the FOO query grammar may force the request-URI to correspond exactly to the search scope.

2.2.2. The Request Body

The server MUST process a text/xml request body, and MAY process request bodies in other formats.

If the client sends a text/xml body, it MUST include the DAV:searchrequest XML element. The DAV:searchrequest XML element identifies the query grammar, defines the criteria, the result record, and any other details needed to perform the search.

2.3. The DAV:searchrequest XML Element

```
<!ELEMENT searchrequest ANY >
```

The DAV:searchrequest XML element contains a single XML element that defines the query. The name of the query element defines the type of the query. The value of that element defines the query itself.

2.4. The Successful 207 (Multistatus) Response

If the server returns 207 (Multistatus), then the search proceeded successfully and the response MUST match that of a PROPFIND.

There MUST be one DAV:response for each resource that matched the search criteria. For each such response, the DAV:href element contains the URI of the resource, and the response MUST include a DAV:propstat element.

In addition, the server MAY include DAV:response items in the reply where the DAV:href element contains a URI that is not a matching resource, e.g. that of a scope or the query arbiter. Each such response item MUST NOT contain a DAV:propstat element, and MUST contain a DAV:status. It SHOULD contain a DAV:responsedescription.

2.4.1. Extending the PROPFIND Response

A response MAY include more information than PROPFIND defines so long as the extra information does not invalidate the PROPFIND response. Query grammars SHOULD define how the response matches the PROPFIND response.

2.4.2. Example: A Simple Request and Response

This example demonstrates the request and response framework. The following XML document shows a simple (hypothetical) natural language query. The name of the query element is FOO:natural-language-query, thus the type of the query is FOO:natural-language-query. The actual query is "Find the locations of good Thai restaurants in Los Angeles". For this hypothetical query, the arbiter returns two properties for each selected resource.

```
SEARCH / HTTP/1.1
Host: ryu.com
Content-Type: text/xml
Connection: Close
Content-Length: 243

<?xml version="1.0"?>
<D:searchrequest xmlns:D = "DAV:" xmlns:F = "FOO:">
  <F:natural-language-query>
    Find the locations of good Thai restaurants in Los Angeles
  </F:natural-language-query>
</D:searchrequest>


>> Response

HTTP/1.1 207 Multi-Status
Content-Type: text/xml
Content-Length: 333

<?xml version="1.0"?>
<D:multistatus xmlns:D="DAV:" xmlns:F="FOO:" xmlns:R="
http://ryu.com/propschema">
  <D:response>
    <D:href>http://siamiam.com</D:href>
    <D:propstat>
      <D:prop>
        <R:location>259 W. Hollywood</R:location>
        <R:rating><R:stars>4</R:stars></R:rating>
      </D:prop>
```

```
        </D:propstat>
      </D:response>
    </D:multistatus>
```

2.5.   Unsuccessful Responses

If an error occurred that prevented execution of the query, the server MUST indicate the failure with the appropriate status code and SHOULD include a DAV:multistatus element to point out errors associated with scopes.

400 Bad Request. The query could not be executed. The request may be malformed (not valid XML for example). Additionally, this can be used for invalid scopes and search redirections.

422 Unprocessable entity. The query could not be executed. If a text/xml request entity was provided, then it may have been valid (well-formed) but may have contained an unsupported or unimplemented query operator.

507 (Insufficient Storage). The query produced more results than the server was willing to transmit. Partial results have been transmitted. The server MUST send a body that matches that for 207, except that there MAY exist resources that matched the search criteria for which no corresponding DAV:response exists in the reply.

2.5.1. Example: Result Set Truncation

A server MAY limit the number of resources in a reply, for example to limit the amount of resources expended in processing a query. If it does so, the reply MUST use status code 507. It SHOULD include the partial results.

When a result set is truncated, there may be many more resources that satisfy the search criteria but that were not examined.

If partial results are included and the client requested an ordered result set in the original request, then any partial results that are returned MUST be ordered as the client directed.

Note that the partial results returned MAY be any subset of the result set that would have satisfied the original query.

```
SEARCH / HTTP/1.1
Host: gdr.com
Content-Type: text/xml
Connection: Close
Content-Length: xxxxx

<?xml version="1.0"?>
<D:searchrequest xmlns:D="DAV:">
  <D:basicsearch>
```

```
      the query goes here
  </D:basicsearch>
</D:searchrequest>


>> Response

HTTP/1.1 507 Insufficient Storage
Content-Type: text/xml
Content-Length: 738
```

```
<?xml version="1.0"?>
<D:multistatus xmlns:D="DAV:">
   <D:response>
      <D:href>http://www.gdr.com/sounds/unbrokenchain.au</D:href>
      <D:propstat>
         <D:prop/>
         <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
   </D:response>
   <D:response>

<D:href>http://tech.mit.edu/archive96/photos/Lesh1.jpg</D:href>
        <D:propstat>
           <D:prop/>
           <D:status>HTTP/1.1 200 OK</D:status>
        <D:/propstat>
   </D:response>
   <D:response>
     <D:href>http://gdr.com</href>
     <D:status>HTTP/1.1 507 Insufficient Storage</D:status>
     <D:responsedescription>
        Only first two matching records were returned
     </D:responsedescription>
   </D:response>
</D:multistatus>
```

2.6. Invalid Scopes & Search Redirections

2.6.1. Indicating an Invalid Scope

A client may submit a scope that the arbiter may be unable to query. The inability to query may be due to network failure, administrative policy, security, etc. This raises the condition described as an "invalid scope".

To indicate an invalid scope, the server MUST respond with a 400 (Bad Request).

The response includes a text/xml body with a DAV:multistatus element. Each DAV:resource in the DAV:multistatus identifies a scope. To indicate that this scope is the source of the error, the server MUST include the DAV:scopeerror element.

2.6.2. Example of an Invalid Scope

```
HTTP/1.1 400 Bad-Request
```

```
        Content-Type: text/xml
        Content-Length: xxxxx

        <?xml version="1.0" ?>

        <d:multistatus xmlns:d="DAV:">
          <d:response>
```

```
    <d:href>http://www.foo.com/X</d:href>
   <d:status>HTTP/1.1 404 Object Not Found</d:status>
    <d:scopeerror/>
  </d:response>
</d:multistatus>
```

2.6.3. Redirections

As described above, a server can indicate only that the scope is invalid. Some search arbiters may be able to indicate that other search arbiters exist for that scope.

In this case, the server MUST:

(1) include the DAV:scopeerror element

(2) include the DAV:status element for that scope. The value of this element MUST be a 303 (See Other) response.

(3) include the DAV:redirectarbiter element for each arbiter the client should use for the redirect. The value of this element is the URI of the arbiter to use. Multiple DAV:redirectarbiter elements are allowed.

2.6.4. Example of a Search Redirection

```
HTTP/1.1 400 Bad-Request
Content-Type: text/xml
Content-Length: xxxxx

<?xml version="1.0" ?>
<?xml:namespace ns="DAV:" prefix="d" ?>

<d:multistatus>
  <d:response>
    <d:href>http://www.foo.com/X</d:href>
              <d:status>HTTP/1.1 303 See Other</d:status>
    <d:scopeerror/>
    <d:redirectarbiter>http://bar.com/B</d:redirectarbiter>
    <d:redirectarbiter>http://baz.com/B</d:redirectarbiter>
  </d:response>
</d:multistatus>
```

2.6.5. Syntax for DAV:scopeerror

```
     <!ELEMENT scopeerror      EMPTY>
```

2.6.6. Syntax for DAV:redirectarbiter

```
     <!ELEMENT redirectarbiter    (#PCDATA)>
```

3. DISCOVERY OF SUPPORTED QUERY GRAMMARS

Servers MUST support discovery of the query grammars supported by a resource.

Clients can determine which query grammars are supported by an arbiter by invoking OPTIONS on the search arbiter. If the resource supports SEARCH, then the DASL response header will appear in the response. The DASL response header lists the supported grammars.

3.1. The OPTIONS Method

The OPTIONS method allows the client to discover if a resource supports the SEARCH method and to determine the list of search grammars supported for that resource.

The client issues the OPTIONS method against a resource named by the Request-URI. This is a normal invocation of OPTIONS defined in [RFC2068].

If a resource supports the SEARCH method, then the server MUST list SEARCH in the OPTIONS response as defined by [RFC2068].

DASL servers MUST include the DASL header in the OPTIONS response. This header identifies the search grammars supported by that resource.

3.2. The DASL Response Header

```
DASLHeader = "DASL" ":" Coded-URL-List
Coded-URL-List : Coded-URL [  ,  Coded-URL-List ]
Coded-URL ; defined in section 8.4 of [WEBDAV]
```

The DASL response header indicates server support for a query grammar in the OPTIONS method. The value is a URI that indicates the type of grammar. This header MAY be repeated.

For example:

```
DASL: <http://foo.bar.com/syntax1>
DASL: <http://akuma.com/syntax2>
DASL: <FOO:natural-language-query>
```

3.3. Example: Grammar Discovery

This example shows that the server supports search on the /somefolder resource with the following query grammars: http://foo.bar.com/syntax1 and http://akuma.com/syntax2.

>> Request

```
OPTIONS /somefolder HTTP/1.1
Connection: Close
Host: ryu.com
```

>> Response

```
HTTP/1.1 200 OK
Date: Tue, 20 Jan 1998 20:52:29 GMT
Connection: close
Accept-Ranges: none
Allow: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, COPY, MOVE,
MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH
Public: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, COPY, MOVE,
MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH
DASL: <http://foo.bar.com/syntax1>
DASL: <http://akuma.com/syntax2>
```

4. QUERY SCHEMA DISCOVERY: QSD

Servers MAY support the discovery of the schema for a query grammar.

The DASL response header provides means for clients to discover the set of query grammars supported by a resource. This alone is not sufficient information for a client to generate a query. For example, the DAV:basicsearch grammar defines a set of queries consisting of a set of operators applied to a set of properties and values, but the grammar itself does not specify which properties may be used in the query.   QSD for the DAV:basicsearch grammar allows a client to discover the set of properties that are searchable, selectable, and sortable. Moreover, although the DAV:basicsearch grammar defines a minimal set of operators, it is possible that a resource might support additional operators in a query. For example, a resource might support a optional operator that can be used to express content-based queries in a proprietary syntax. QSD allows a client to discover these operators and their syntax. The set of discoverable quantities will differ from grammar to grammar, but each grammar can define a means for a client to discover what can be discovered.

In general, the schema for a given query grammar depends on both the resource (the arbiter) and the scope. A given resource might have access to one set of properties for one potential scope, and another set for a different scope. For example, consider a server able to search two distinct collections, one holding cooking recipes, the other design documents for nuclear weapons. While both collections might support properties such as author, title, and date, the first might also define properties such as calories and preparation time, while the second defined properties such as yield and applicable patents. Two distinct arbiters indexing the same collection might also have access to different properties. For example, the recipe collection mentioned above might also indexed by a value-added server that also stored the names of chefs who had tested the recipe. Note also that the available query schema might also depend on other factors, such as the identity of the principal conducting the search, but these factors are not exposed in this protocol.

Each query grammar supported by DASL defines its own syntax for expressing the possible query schema. A client retrieves the schema for a given query grammar on an arbiter resource with a given scope by invoking the SEARCH method on that arbiter, with that grammar and scope, with a query whose DAV:select element includes the DAV:queryschema property. This property is defined only in the context of such a search, a server SHOULD not treat it as defined in the context of a PROPFIND on the scope. The content of this property is an XML element whose name and syntax depend upon the grammar, and whose value may (and likely will) vary depending upon the grammar, arbiter, and scope.

The query schema for DAV:basicsearch is defined in section 5.19.

4.1. The DAV:queryschema Property

```
<!ELEMENT queryschema ANY >
```

4.1.1. Example of query schema discovery

In this example, the arbiter is recipes.com, the grammar is DAV:basicsearch, the scope is also recipes.com.

```
SEARCH / HTTP/1.1
Host: recipes.com
Content-Type: application/xml
Connection: Close
Content-Length: xxx

<?xml version="1.0"?>
<D:searchrequest xmlns:D="DAV:">
  <D:basicsearch>
    <D:select>
       <D:queryschema/>
    </D:select>

<D:from><D:scope><D:href>http://recipes.com</d:href></D:scope></D
:from>
  </D:basicsearch>
</D:searchrequest>
```

Response:

```
HTTP/1.1 207 Multistatus
Content-Type: application/xml
Content-Length: xxx
```

```
        <?xml version="1.0"?>
        <D:multistatus xmlns:D="DAV:">
          <D:response>
            <D:href>http://recipes.com</D:href>
            <D:propstat>
               <D:prop>
                 <D:querygrammar>
                   <D:basicsearchschema>
```

```
                  See section 5.19.9 for actual contents
                </D:basicsearchschema>
              </D:querygrammar>
            </D:prop>
            <D:status>HTTP/1.1 200 Okay</D:status>
         </D:propstat>
       </D:response>
     </D:multistatus>
```

5. THE DAV:BASICSEARCH GRAMMAR

5.1.   Introduction

DAV:basicsearch uses an extensible XML syntax that allows clients to express search requests that are generally useful for WebDAV scenarios. DASL-extended servers MUST accept this grammar, and MAY accept others grammars.

DAV:basicsearch has several major components: DAV:select, DAV:from, DAV:where, DAV:orderby, and DAV:limit. DAV:select provides the result record definition. DAV:from defines the scope. DAV:where defines the criteria. DAV:orderby defines the sort order of the result set. DAV:limit provides constraints on the query as a whole.

5.2.   The DAV:basicsearch DTD

```
<!ELEMENT basicsearch   (select, from, where?, orderby?, limit?)
>

<!ELEMENT select       (allprop | prop) >

<!ELEMENT from         (scope) >
<!ELEMENT scope        (href, depth?) >


<!ENTITY %comp_ops      "eq | lt | gt| lte | gte">
<!ENTITY %log_ops "and | or | not">
<!ENTITY %special_ops   "isdefined">
<!ENTITY %string_ops      "like">
<!ENTITY %content_ops   "contains">

<!ENTITY %all_ops "%comp_ops; | %log_ops; | %special_ops;
|
                  %string_ops; | %content_ops;">
```

```
        <!ELEMENT where        ( %all_ops; ) >

        <!ELEMENT and          ( ( %all_ops; ) +) >

        <!ELEMENT or ( ( %all_ops; ) +) >
```

```
<!ELEMENT not          ( %all_ops; ) >

<!ELEMENT lt ( prop , literal ) >
<!ATTLIST lt casesensitive
  (1|0) 1 >

<!ELEMENT lte          ( prop , literal ) >
<!ATTLIST lte          casesensitive
  (1|0) 1 >

<!ELEMENT gt ( prop , literal) >
<!ATTLIST gt casesensitive
  (1|0) 1 >

<!ELEMENT gte          ( prop , literal ) >
<!ATTLIST gte          casesensitive
  (1|0) 1 >

<!ELEMENT eq ( prop , literal ) >
<!ATTLIST eq casesensitive
  (1|0) 1 >

<!ELEMENT literal (#PCDATA)>
<!ATTLIST literal xml:space (default|preserve) preserve >

<!ELEMENT isdefined       (prop) >
<!ELEMENT like           (prop, literal) >
<!ELEMENT contains      (#PCDATA)>

<!ELEMENT orderby       (order+) >
<!ELEMENT order        (prop, (ascending | descending)?)
<!ATTLIST order        casesensitive
  (1|0) 1 >
<!ELEMENT ascending     EMPTY>
<!ELEMENT descending      EMPTY>

<!ELEMENT limit        (nresults) >
<!ELEMENT nresults      (#PCDATA) >
```

5.2.1. Example Query

This query retrieves the content length values for all resources located under the server's "/container1/" URI namespace whose length exceeds 10000.

```
<d:searchrequest>
```

```
        <d:basicsearch>
          <d:select>
              <d:prop><d:getcontentlength/></d:prop>
          </d:select>
          <d:from>
            <d:scope>
              <d:href>/container1/</d:href>
              <d:depth>infinity</d:depth>
            </d:scope>
          </d:from>
          <d:where>
            <d:gt>
              <d:prop><d:getcontentlength/></d:prop>
              <d:literal>10000</d:literal>
            </d:gt>
          </d:where>
          <d:orderby>
            <d:order>
                    <d:prop><d:getcontentlength/><d:prop>
                    <d:ascending/>
            </d:order>
          </d:orderby>
        </d:basicsearch>
        </d:searchrequest>
```

5.3.   DAV:select

DAV:select defines the result record. This document defines two possible values: DAV:allprop and DAV:prop, both defined in [WebDAV].

If the value is DAV:allprop, the result record for a given resource includes all the properties for that resource.

If the value is DAV:prop, then the result record for a given resource includes only those properties named by the DAV:prop element. Each property named by the DAV:prop element must be referenced in the Multistatus response.

The rules governing the status codes for each property match those of the PROPFIND method defined in [WebDAV].

5.4.   DAV:from

DAV:from defines the query scope. This contains exactly one DAV:scope element. The scope element contains a mandatory DAV:href element and an optional DAV:depth element.

DAV:href indicates the URI for a collection to use as a scope.

When the scope is a collection, if DAV:depth is "1", the search includes the members of the collection. When it is "infinity", the search includes all recursive members of the collection.8.5.1.

5.4.1. Relationship to the Request-URI

If the DAV:scope element is an absolute URI, the scope is exactly that URI.

If the DAV:scope element is a relative URI, the scope is taken to be relative to the request-URI.

5.4.2. Scope

A Scope can be an arbitrary URI.

Servers, of course, may support only particular scopes. This may include limitations for particular schemes such as "http:" or "ftp:" or certain URI namespaces.

If a scope is given that is not supported the server MUST respond with a 400 status code that includes a Multistatus error. A scope in the query appears as a resource in the response and must include an appropriate status code indicating its validity with respect to the search arbiter.

Example:

```
HTTP/1.1 400 Bad Request
Content-Type: text/xml
Content-Length: 428

<?xml version="1.0" ?>
<d:multistatus xmlns:D="DAV:" xmlns:F="FOO:" >

  <d:response>
    <d:href>http://www.foo.com/scope1</d:href>
    <d:status>HTTP/1.1 502 Bad Gateway</d:status>
  </d:response>

</d:multistatus>
```

This example shows the response if there is a scope error. The response provides a Multistatus with a status for the scope. In this case, the scope cannot be reached because the server cannot search another server (502).

5.5. DAV:where

DAV:where element defines the search condition for inclusion of resources in the result set. The value of this element is an XML element that defines a search operator that evaluates to one of the Boolean truth values TRUE, FALSE, or UNKNOWN. The search operator contained by DAV:where may itself contain and evaluate additional search operators as operands, which in turn may contain and evaluate additional search operators as operands, etc. recursively.

5.5.1. Use of Three-Valued Logic in Queries

Each operator defined for use in the where clause that returns a Boolean value MUST evaluate to TRUE, FALSE, or UNKNOWN. The resource under scan is included as a member of the result set if and only if the search condition evaluates to TRUE.

Consult Appendix A for details on the application of three-valued logic in query expressions.

5.5.2. Handling Optional operators

If a query provides an operator that is not supported by the server, then the server MUST respond with a 422 (Unprocessable Entity) status code.

5.5.3. Treatment of NULL Values

If a SEARCH PROPFIND for a property value would yield a 404 or 403 response for that property, then that property is considered NULL.

NULL values are "less than" all other values in comparisons.

Empty strings (zero length strings) are not NULL values. An empty string is "less then" a string with length greater than zero.

The DAV:isdefined operator is defined to test if the value of a property is NULL.

5.5.4. Example: Testing for Equality

The example shows a single operator (DAV:eq) applied in the criteria.

```
<d:where>
  <d:eq>
    <d:prop> <d:getcontentlength/> </d:prop>
    <d:literal> 100 </d:literal>
  </d:eq>
</d:where>
```

5.5.5. Example: Relative Comparisons

The example shows a more complex operation involving several operators (DAV:and, DAV:eq, DAV:gt) applied in the criteria. This DAV:where expression matches those resources that are "image/gifs" over 4K in size.

```
<D:where>
  <D:and>
    <D:eq>
      <D:prop> <D:getcontenttype/> </D:prop>
      <D:literal> image/gif </D:literal>
    </D:eq>
```

```
            <D:gt>
              <D:prop> <D:getcontentlength/> </D:prop>
              <D:literal> 4096 </D:literal>
            </D:gt>
          </D:and>
        </D:where>
```

5.6. DAV:orderby

The DAV:orderby element specifies the ordering of the result set. It contains one or more DAV:order elements, each of which specifies a comparison between two items in the result set. Informally, a comparison specifies a test that determines whether one resource appears before another in the result set. Comparisons are applied in the order they occur in the DAV:orderby element, earlier comparisons being more significant.

The comparisons defined here use only a single property from each resource, compared using the same ordering as the DAV:lt operator (ascending) or DAV:gt operator (descending). If neither direction is specified, the default is DAV:ascending.

In the context of the DAV:orderby element, null values are considered to collate before any actual (i.e., non null) value, including strings of zero length (as in ANSI standard SQL, c.f., ANSI X3.135-1992).

5.6.1. Comparing Natural Language Strings.

Comparisons on strings take into account the language defined for that property. Clients MAY specify the language using the xml:lang attribute. If no language is specified either by the client or defined for that property by the server or if a comparison is performed on strings of two different languages, the results are undefined.

The DAV:casesensitive attribute may be used to indicate case-sensitivity for comparisons.

5.6.2. Example of Sorting

This sort orders first by last name of the author, and then by size, in descending order, so that the largest works appear first.

```
<d:orderby>
  <d:order>
     <d:prop><r:lastname/></d:prop>
     <d:ascending/>
  </d:order>
  <d:order>
     <d:prop><d:getcontentlength/></d:prop>
     <d:descending/>
  </d:order>
</d:orderby>
```

5.7. Boolean Operators: DAV:and, DAV:or, and DAV:not

The DAV:and operator performs a logical AND operation on the expressions it contains.

The DAV:or operator performs a logical OR operation on the values it contains.

The DAV:not operator performs a logical NOT operation on the values it contains.

5.8. DAV:eq

The DAV:eq operator provides simple equality matching on property values.

The DAV:casesensitive attribute may be used with this element.

5.9. DAV:lt, DAV:lte, DAV:gt, DAV:gte

The DAV:lt, DAV:lte, DAV:gt, and DAV:gte operators provide comparisons on property values. The DAV:casesensitive attribute may be used with these elements.

5.10. DAV:literal

DAV:literal allows literal values to be placed in an expression.

Because white space in literal values is significant to in comparisons, DAV:literal makes use of the xml:space attribute to identify this significance. The default value of this attribute for DAV:literal is preserve. Consult section 2.10 of [XML] for more information on the use of this attribute.

5.11. DAV:isdefined

The DAV:isdefined operator allows clients to determine whether a property is defined on a resource on a resource. The meaning of "defined on a resource" is found in section 5.5.3.

Example:

```
<d:isdefined>
  <d:prop><x:someprop/></d:prop>
</d:isdefined>
```

The DAV:isdefined operator is optional.

5.12. DAV:like

The DAV:like is an optional operator intended to give simple wildcard-based pattern matching ability to clients.

The operator takes two arguments.

The first argument is a DAV:prop element identifying a single property to evaluate.

The second argument is a DAV:literal element that gives the pattern matching string.

5.12.1. Syntax for the Literal Pattern

```
Pattern := [wildcard] 0*( text [wildcard] )
wildcard := exactlyone | zeroormore
text := 1*( <octet> | escapesequence )
exactlyone : = "?"
zeroormore := "%"
escapechar := "\"
escapesequence := "\" ( exactlyone | zeroormore | escapechar )
```

The value for the literal is composed of wildcards separated by segments of text. Wildcards may begin or end the literal. Wildcards may not be adjacent.

The "?" wildcard matches exactly one character.

The "%" wildcard matches zero or more characters

The "\" character is an escape sequence so that the literal can include "?" and "%". To include the "\" character in the pattern, the escape sequence "\\" is used..

5.12.2. Example of DAV:like

This example shows how a client might use DAV:like to identify those resources whose content type was a subtype of image.

```
<D:where>
  <D:like>
    <D:prop><D:getcontenttype/></D:prop>
    <D:literal>image%</D:literal>
  </D:like>
</D:where>
```

5.13. DAV:contains

The DAV:contains operator is an optional operator that provides content-based search capability. This operator implicitly searches against the text content of a resource, not against content of properties. The DAV:contains operator is intentionally not overly constrained, in order to allow the server to do the best job it can in performing the search.

The DAV:contains operator evaluates to a Boolean value. It evaluates to TRUE if the content of the resource satisfies the search. Otherwise, It evaluates to FALSE.

Within the DAV:contains XML element, the client provides a phrase: a single word or whitespace delimited sequence of words. Servers MAY ignore punctuation in a phrase. Case-sensitivity is left to the server.

The following things may or may not be done as part of the search: Phonetic methods such as soundex may or may not be used. Word stemming may or may not be performed. Thesaurus expansion of words may or may not be done. Right or left truncation may or may not be performed. The search may be case insensitive or case sensitive. The word or words may or may not be interpreted as names. Multiple words may or may not be required to be adjacent or "near" each other. Multiple words may or may not be required to occur in the same order. Multiple words may or may not be treated as a phrase. The search may or may not be interpreted as a request to find documents "similar" to the string operand.

The DAV:score property is intended to be useful to rank documents satisfying the DAV:Contains operator.

5.13.1. Example

The example below shows a search for the phrase Peter Forsberg .

Depending on its support for content-based searching, a server MAY treat this as a search for documents that contain the words Peter and Forsberg .

```
<D:where>
  <D:contains>Peter Forsberg</D:contains>
</D:where>
```

5.13.2. Example

The example below shows a search for resources that contain Peter and Forsberg .

```
<D:where>
  <D:and>
    <D:contains>Peter</D:contains>
    <D:contains>Forsberg</D:contains>
  </D:and>
</D:where>
```

5.14. The DAV:limit XML Element

```
<!ELEMENT limit (nresults) >
```

The DAV:limit XML element contains requested limits from the client to limit the size of the reply or amount of effort expended by the server.

5.15. The DAV:nresults XML Element

```
<!ELEMENT nresults (#PCDATA)> ;only digits
```

The DAV:nresults XML element contains a requested maximum number of records to be returned in a reply. The server MAY disregard this limit. The value of this element is an integer.

5.16. The DAV:casesensitive XML attribute

The DAV:casesensitive attribute allows clients to specify case-sensitive or case-insensitive behavior for DAV:basicsearch operators.

The possible values for DAV:casesensitive are "1" or "0". The "1" value indicates case-sensitivity. The "0" value indicates case-insensitivity. The default value is server-specified.

Support for the DAV:casesensitive is optional. A server should respond with an error 422 if the DAV:casesensitive attribute is used but cannot be supported.

5.17. The DAV:score Property

```
<!ELEMENT score  (#PCDATA)>
```

The DAV:score XML element is a synthetic property whose value is defined only in the context of a query result where the server computes a score, e.g. based on relevance. It may be used in DAV:select or DAV:orderby elements. Servers SHOULD support this property. The value is a string representing the score, an integer from zero to 10000 inclusive, where a higher value indicates a higher score (e.g. more relevant).

Clients should note that, in general, it is not meaningful to compare the numeric values of scores from two different queries unless both were executed by the same underlying search system on the same collection of resources.

5.18. The DAV:iscollection Property

```
<!ELEMENT iscollection (#PCDATA)>
```

The DAV:iscollection XML element is a synthetic property whose value is defined only in the context of a query.

The property is TRUE (the literal string "1") of a resource if and only if a PROPFIND of the DAV:resourcetype property for that resource would contain the DAV:collection XML element. The property is FALSE (the literal string "0") otherwise.

Rationale: This property is provided in lieu of defining generic structure queries, which would suffice for this and for many more powerful queries, but seems inappropriate to standardize at this time.

5.18.1. Exampe of DAV:iscollection

This example shows a search criterion that picks out all and only the resources in the scope that are collections.

```
<D:where>
  <D:eq>
```

```
        <D:prop><D:iscollection></D:prop>
        <D:literal>1<D:literal>
      </D:eq>
    </D:where>
```

5.19. Query Schema for DAV:basicsearch

The DAV:basicsearch grammar defines a search criteria that is a Boolean-valued expression, and allows for an arbitrary set of properties to be included in the result record. The result set may be sorted on a set of property values. Accordingly the DTD for schema discovery for this grammar allows the server to express:

- the set of properties that may be either searched, returned, or used to sort, and a hint about the data type of such properties

- the set of optional operators defined by the resource.

5.19.1. DTD for DAV:basicsearch QSD

```
<!ELEMENT basicsearchschema (properties, operators)>
<!ELEMENT properties    (propdesc*)>
<!ELEMENT propdesc    (prop, ANY)>
<!ELEMENT operators   (opdesc*)>
<!ELEMENT opdesc        ANY>
<!ELEMENT operand_property      EMPTY>
<!ELEMENT operand_literal       EMPTY>
```

The DAV:properties element holds a list of descriptions of properties.

The DAV:operators element describes the optional operators that may be used in a DAV:where element.

5.19.2. DAV:propdesc Element

Each instance of a DAV:propdesc element describes the property or properties in the DAV:prop element it contains. All subsequent elements are descriptions that apply to those properties. All descriptions are optional and may appear in any order. Servers SHOULD support all the descriptions defined here, and MAY define others.

DASL defines eight descriptions. The first group (DAV:datatype, DAV:searchable, DAV:selectable, DAV:sortable, and

DAV:casesensitive) provide a hints about the property value, and may be useful to a user interface prompting for a value. The second group identify portions of the query (DAV:where, DAV:select, and DAV:orderby). If a property has a description for a section, then the server MUST allow the property to be used in that section. These descriptions are optional. If a property does not have such a description, or is not described at all, then the server MAY still allow the property to be used in the corresponding section.

5.19.3. The DAV:datatype Property Description

The DAV:datatype element contains a single XML element that provides a hint about the domain of the property, which may be useful to a user interface prompting for a value to be used in a query. The namespace for expressing a DASL defined data type is "urn:uuid:C2F41010-65B3-11d1-A29F-00AA00C14882/".

```
<!ELEMENT datatype        ANY >
```

DASL defines the following data type elements:

NAME	CONTENTS EXAMPLE
boolean	1 0
string	Foobar
int	-259 23
float	.314159265358979E+1 5.33
dateTime.tz	1994-11-05T08:15:5Z

If the data type of a property is not given, then the data type defaults to string.

5.19.4. The DAV:searchable Property Description

```
<!ELEMENT searchable      EMPTY >
```

If this element is present, then the server MUST allow this property to appear within a DAV:where element where an operator allows a property. Allowing a search does not mean that the property is guaranteed to be defined on every resource in the scope, it only indicates the server's willingness to check.

5.19.5. The DAV:selectable Property Description

```
<!ELEMENT selectable      EMPTY >
```

This element indicates that the property may appear in the
DAV:select element.

5.19.6. The DAV:sortable Property Description

This element indicates that the property may appear in the DAV:orderby element

```
<!ELEMENT sortable        EMPTY >
```

5.19.7. The DAV:casesensitive Property Description

This element only applies to properties whose data type is "string" as per the DAV:datatype property description. Its presence indicates that compares performed for searches, and the comparisons for ordering results on the string property will be case sensitive. (The default is case insensitive.)

```
<!ELEMENT casesensitive EMPTY >
```

5.19.8. The DAV:operators XML Element

The DAV:operators element describes every optional operator supported in a query. (Mandatory operators are not listed since they are mandatory and permit no variation in syntax.). All optional operators that are supported MUST be listed in the DAV:operators element. The listing for an operator consists of the operator (as an empty element), followed by one element for each operand. The operand MUST be either DAV:operand_property or DAV:operand_literal, which indicate that the operand in the corresponding position is a property or a literal value, respectively. If an operator is polymorphic (allows more than one operand syntax) then each permitted syntax MUST be listed separately.

```
<D:propdesc><D:like/><D:operand_property/><D:operand_literal/></D
:propdesc>
```

5.19.9. Example of Query Schema for DAV:basicsearch

```
<D:basicsearchschema xmlns:D="DAV:" xmlns:t="urn:uuid:C2F41010-
65B3-11d1-A29F-00AA00C14882/" xmlns:J="http://jennicam.org">
  <D:properties>
    <D:propdesc>
      <D:prop><D:getcontentlength/></D:prop>
      <D:datatype><t:int></D:datatype>
      <D:searchable/><D:selectable/><D:sortable/>
    </D:propdesc>
    <D:propdesc>
      <D:prop><D:getcontenttype/><D:displayname></D:prop>
```

```
            <D:searchable/><D:selectable/> <D:sortable/>
          </D:propdesc>
          <D:propdesc>
            <D:prop><J:fstop/></D:prop>
            <D:selectable/>
          </D:propdesc>
        </D:properties>
```

```
    <D:operators>
      <D:opdesc>
      <D:isdefined/><D:operand_property/>
    </D:opdesc>
     <D:opdesc>
      <D:like/><D:operand_property/><D:operand_literal/>
    </D:opdesc>
    </D:operators>
  </D:basicsearchschema>
```

This response lists four properties. The datatype of the last three properties is not given, so it defaults to string. All are selectable, and the first three may be searched. All but the last may be used in a sort. Of the optional DAV operators, DAV:isdefined and DAV:like are supported.

Note: The schema discovery defined here does not provide for discovery of supported values of the DAV:casesensitive attribute. This may require that the reply also list the mandatory operators.

6. INTERNATIONALIZATION CONSIDERATIONS

Clients have the opportunity to tag properties when they are stored in a language. The server SHOULD read this language-tagging by examining the xml:lang attribute on any properties stored on a resource.

The xml:lang attribute specifies a nationalized collation sequence when properties are compared.

Comparisons when this attribute differs have undefined order.

7. SECURITY CONSIDERATIONS

This section is provided to detail issues concerning security implications of which DASL applications need to be aware. All of the security considerations of HTTP/1.1 also apply to DASL. In addition, this section will include security risks inherent in searching and retrieval of resource properties and content.

A query must not allow one to retrieve information about values or existence of properties that one could not obtain via PROPFIND. (e.g. by use in DAV:orderby, or in expressions on properties.)

Server should prepare for denial of service attacks. For example a client may issue a query for which the result set is expensive to calculate or transmit because many resources match or must be evaluated.

8. SCALABILITY

Query grammars are identified by URIs. Applications SHOULD not attempt to retrieve these URIs even if they appear to be retrievable (for example, those that begin with "http://")

9. AUTHENTICATION

Authentication mechanisms defined in WebDAV will also apply to DASL.

10. IANA CONSIDERATIONS

This document uses the namespace defined by [WebDAV] for XML elements. All other IANA considerations mentioned in [WebDAV] also applicable to DASL

11. COPYRIGHT

To be supplied.

12. INTELLECTUAL PROPERTY

To be supplied.

13. REFERENCES

[DASLREQ] J. Davis, S. Reddy, J. Slein, "Requirements for DAV Searching and Locating", September 3 1998, internet-draft, work-in-progress, draft-davis-dasl-requirements-00.txt

[RFC2068] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, U.C. Irvine, DEC, MIT/LCS, January 1997.

[RFC2119] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels." RFC 2119, BCP 14. Harvard University. March, 1997.

[WebDAV] Y. Goland, E.J. Whitehead, A. Faizi, S.R. Carter, D. Jenson, "Extensions for Distributed Authoring on the World Wide Web", November 16 1998, internet-draft, work-in-progress, draft-ietf-webdav-protocol-10.txt.

[XML] T. Bray, J. Paoli, C. M. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0", September 16, 1998, W3C Recommendation.

[XMLNS] T. Bray, D. Hollander, A. Layman, "Namespaces in XML", November, 1998, W3C Proposed Recommendation. http://www.w3.org/TR/PR-xml-names

14. AUTHOR'S ADDRESSES

Saveen Reddy
Microsoft
One Microsoft Way
Redmond WA, 9085-6933
Email:saveenr@microsoft.com

Dale Lowry
Novell
1555 N. Technology Way
M/S ORM-M-314
Orem, UT 84097
Email: dlowry@novell.com

Surendra Reddy
Oracle Corporation
600 Oracle Parkway, M/S 6op3,
Redwoodshores, CA 94065
Email: skreddy@us.oracle.com
Phone:(650) 506 5441

Rick Henderson
Netscape
Email: rickh@netscape.com

Jim Davis
Xerox PARC
3333 Coyote Hill Road
Palo Alto CA 94304
650-812-4301
Email: jdavis@parc.xerox.com

Alan Babich
Filenet
3565 Harbor Blvd.
Costa Mesa, CA 92626
714-966-3403
Email: ababich@filenet.com

15. APPENDICES

Appendix A Three-Valued Logic in DAV:basicsearch

ANSI standard three valued logic is used when evaluating the search condition (as defined in the ANSI standard SQL specifications, for example in ANSI X3.135-1992, section 8.12, pp. 188-189, section 8.2, p. 169, General Rule 1)a), etc.).

ANSI standard three valued logic is undoubtedly the most widely practiced method of dealing with the issues of properties in the search condition not having a value (e.g., being null or not defined) for the resource under scan, and with undefined expressions in the search condition (e.g., division by zero, etc.). Three valued logic works as follows.

Undefined expressions are expressions for which the value of the expression is not defined. Undefined expressions are a completely separate concept from the truth value UNKNOWN, which is, in fact, well defined. Property names and literal constants are considered expressions for purposes of this section. If a property in the current resource under scan has not been set to a value (either because the property is not defined for the current resource, or because it is null for the current resource), then the value of that property is undefined for the resource under scan. DASL 1.0 has no arithmetic division operator, but if it did, division by zero would be an undefined arithmetic expression.

If any subpart of an arithmetic, string, or datetime subexpression is undefined, the whole arithmetic, string, or datetime subexpression is undefined.

There are no manifest constants to explicitly represent undefined number, string, or datetime values.

Since a Boolean value is ultimately returned by the search condition, arithmetic, string, and datetime expressions are always arguments to other operators. Examples of operators that convert arithmetic, string, and datetime expressions to Boolean values are the six relational operators ("greater than", "less than", "equals", etc.). If either or both operands of a relational operator have undefined values, then the relational operator evaluates to UNKNOWN. Otherwise, the relational operator evaluates to TRUE or FALSE, depending upon the outcome of the comparison.

The Boolean operators DAV:and, DAV:or and DAV:not are evaluated according to the following rules:

```
  UNKNOWN and UNKNOWN =         UNKNOWN

  UNKNOWN or UNKKNOWN =         UNKNOWN

  not UNKNOWN =                 UNKNOWN



  UNKNOWN and TRUE =            UNKNOWN

  UNKNOWN and FALSE =           FALSE
```

UNKNOWN and UNKNOWN =	UNKNOWN
UNKNOWN or TRUE =	TRUE
UNKNOWN or FALSE =	UNKNOWN
UNKNOWN or UNKNOWN =	UNKNOWN

16. CHANGE HISTORY

Feb 14, 1998

Initial Draft

Feb 28, 1998

Referring to DASL as an extension to HTTP/1.1 rather than DAV

Added new sections "Notational Conventions", "Protocol Model", "Security Considerations"

Changed section 3 to "Elements of Protocol"

Added some stuff to introduction

Added "result set" terminology

Added "IANA Considerations".

Mar 9, 1998

Moved sub-headings of "Elements of Protocol" to first level and removed "Elements of Protocol" Heading.

Added an sentence in introduction explaining that this is a "sketch" of a protocol.

Mar 11, 1998

Added orderby, data typing, three valued logic, query schema property, and element definitions for schema for basicsearch.

April 8, 1998

- made changes based on last week s DASL BOF.

May 8, 1998

Removed most of DAV:searcherror; converted to DAV:searchredirect

Altered DAV:basicsearch grammar to use avoid use of ANY in DTD

June 17, 1998

-Added details on Query Schema Discovery

-Shortened list of data types

June 23, 1998

moved data types before change history

rewrote the data types section

removed the casesensitive element and replace with the casesensitive attribute

added the casesensitive attribute to the DTD for all operations that might work on a string

Jul 20, 1998

A series of changes. See Author s meeting minutes for details.

July 28, 1998

Changes as per author's meeting. QSD uses SEARCH, not PROPFIND.

Moved text around to keep concepts nearby.

Boolean literals are 1 and 0, not T and F.

contains changed to contentspassthrough.

Renamed rank to score.

July 28, 1998

Added Dale Lowry as Author

September 4, 1998

Added 422 as response when query lists unimplemented operators.

dav:literal declares a default value for xml:space, 'preserve' (see XML spec, section 2.10)

moved to new XML namespace syntax

September 22, 1998

Changed "simplesearch" to "basicsearch"

Changed isnull to isdefined

Defined NULLness as having a 404 or 403 response used ENTITY syntax in DTD

Added redirect

October 9, 1998

Fixed a series of typographical and formatting errors.

Modified the section of three-valued logic to use a table rather than a text description of the role of UNKNOWN in expressions.

November 2, 1998

Added the DAV:contains operator.

Removed the DAV:contentpassthrough operator.

November 18, 1998

Various author comments for submission