Network Working Group
Request for Comments: 5323
Category: Standards Track

J. Reschke, Editor
greenbytes
S. Reddy
Mitrix
J. Davis
A. Babich
IBM
November 2008

Web Distributed Authoring and Versioning (WebDAV) SEARCH

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document specifies a set of methods, headers, and properties composing Web Distributed Authoring and Versioning (WebDAV) SEARCH, an application of the HTTP/1.1 protocol to efficiently search for DAV resources based upon a set of client-supplied criteria.

[1] http://trustee.ietf.org/license-info

Table of Contents

1. Introduction

1.1. DASL

This document defines Web Distributed Authoring and Versioning (WebDAV) SEARCH, an application of HTTP/1.1 forming a lightweight search protocol to transport queries and result sets that allows clients to make use of server-side search facilities. It is based on earlier work done in the IETF DASL Working Group (see Section 10). In this specification, the terms "WebDAV SEARCH" and "DASL" are used interchangeably.

DASL minimizes 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 and the request/response formats defined for it (Section 2),
- feature discovery through the "DASL" response header and the optional DAV:supported-grammar-set property (Section 3),
- optional grammar schema discovery (Section 4), and
- one mandatory grammar: DAV:basicsearch (Section 5).

1.2. Relationship to DAV

DASL relies on the resource and property model defined by [RFC4918]. DASL does not alter this model. Instead, DASL allows clients to access DAV-modeled resources through server-side search.

1.3. Terms

This document uses the terms defined in [RFC2616], [RFC4918], [RFC3253], and in this section.

Criteria

An expression against which each resource in the search scope is evaluated.

Query

A query is a combination of a search scope, search criteria, result record definition, sort specification, and a search modifier.

Query Grammar

A set of definitions of XML elements, attributes, and constraints on their relations and values that defines a set of queries and the intended semantics.

Query Schema

A listing, for any given grammar and scope, of the properties and operators that may be used in a query with that grammar and scope.

Result

A result is a result set, optionally augmented with other information describing the search as a whole.

Result Record

A description of a resource. A result record is a set of properties, and possibly other descriptive information.

Result Record Definition

A specification of the set of properties to be returned in the result record.

Result Set

A set of records, one for each resource for which the search criteria evaluated to True.

Scope

A set of resources to be searched.

Search Arbiter

A resource that supports the SEARCH method.

Search Modifier

An instruction that governs the execution of the query but is not part of the search scope, result record definition, the search criteria, or the sort specification. An example of a search modifier is one that controls how much time the server can spend on the query before giving a response.

Sort Specification

A specification of an ordering on the result records in the result set.

1.4. Notational Conventions

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234], unless explicitly stated otherwise.

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].

This document uses XML DTD fragments ([XML], Section 3.2) as a purely notational convention. WebDAV request and response bodies cannot be validated by a DTD due to the specific extensibility rules defined in Section 17 of [RFC4918] and due to the fact that all XML elements defined by this specification use the XML namespace name "DAV:". In particular:

1. element names use the "DAV:" namespace,
2. element ordering is irrelevant unless explicitly stated,
3. extension elements (elements not already defined as valid child elements) may be added anywhere, except when explicitly stated otherwise,
4. extension attributes (attributes not already defined as valid for this element) may be added anywhere, except when explicitly stated otherwise.

When an XML element type in the "DAV:" namespace is referenced in this document outside of the context of an XML fragment, the string "DAV:" will be prefixed to the element type.

Similarly, when an XML element type in the namespace "http://www.w3.org/2001/XMLSchema" is referenced in this document outside of the context of an XML fragment, the string "xs:" will be prefixed to the element type.

This document inherits, and sometimes extends, DTD productions from Section 14 of [RFC4918].

1.5. Note on Usage of 'DAV:' XML Namespace

This specification defines elements, properties, and condition names in the XML namespace "DAV:". In general, only specifications authored by IETF working groups are supposed to do this. In this case an exception was made, because WebDAV SEARCH started its life in the IETF DASL working group (<http://www.webdav.org/dasl/>, and at the time the working group closed down there was already significant deployment of this specification.

1.6. 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 or application/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 an entity that matches the WebDAV multistatus format ([RFC4918], Section 13).

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 an entity matching the WebDAV multistatus format ([RFC4918], Section 13).

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.

SEARCH is a safe method; it does not have any significance other than executing a query and returning a query result (see [RFC2616], Section 9.1.1).

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. Any HTTP resource may function as search arbiter. It is not a new type of resource (in the sense of DAV:resourcetype as defined in [RFC4918], Section 15.9), nor does it have to be a WebDAV-compliant resource.

The SEARCH method 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, a 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 or application/xml request body, and MAY process request bodies in other formats. See [RFC3023] for guidance on packaging XML in requests.

Marshalling:

If a request body with content type text/xml or application/xml is included, it MUST be either a DAV:searchrequest or a DAV:query-schema-discovery XML element. Its single child element identifies the query grammar.

For DAV:searchrequest, the definition of search criteria, the result record, and any other details needed to perform the search depend on the individual search grammar.

For DAV:query-schema-discovery, the semantics is defined in Section 4.

Preconditions:

(DAV:search-grammar-discovery-supported): when an XML request body is present and has a DAV:query-schema-discovery document element, the server MUST support the query schema discovery mechanism described in Section 4.

(DAV:search-grammar-supported): when an XML request body is present, the search grammar identified by the document element's child element must be a supported search grammar.

(DAV:search-multiple-scope-supported): if the SEARCH request specified multiple scopes, the server MUST support this optional feature.

(DAV:search-scope-valid): the supplied search scope must be valid. There can be various reasons for a search scope to be invalid, including unsupported URI schemes and communication problems. Servers MAY add [RFC4918] compliant DAV:response elements as content to the condition element indicating the precise reason for the failure.

2.3. The Successful 207 (Multistatus) Response

If the server returns 207 (Multistatus), then the search proceeded successfully, and the response MUST use the WebDAV multistatus format ([RFC4918], Section 13). The results of this method SHOULD NOT be cached.

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.

> **Note:** the WebDAV multistatus format requires at least one DAV:response child element. This specification relaxes that restriction so that empty results can be represented.

Note that for each matching resource found, there may be multiple URIs within the search scope mapped to it. In this case, a server SHOULD report only one of these URIs. Clients can use the live property DAV:resource-id, defined in Section 3.1 of [WEBDAV-BIND] to identify possible duplicates.

2.3.1. 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 207, return a DAV:multistatus response body, and indicate a status of 507 (Insufficient Storage) for the search arbiter URI. 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.

2.3.2. 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.

2.3.3. 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 natural-language-query in the XML namespace "http://example.com/foo". 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.

>> Request:

```
SEARCH / HTTP/1.1
Host: example.org
Content-Type: application/xml; charset="utf-8"
Content-Length: 252

<?xml version="1.0" encoding="UTF-8"?>
<D:searchrequest xmlns:D="DAV:" xmlns:F="http://example.com/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; charset="utf-8"
Content-Length: 429

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:"
   xmlns:R="http://example.org/propschema">
  <D:response>
    <D:href>http://siamiam.example/</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:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

2.3.4. Example: Result Set Truncation

In the example below, the server returns just two results, and then indicates that the result is truncated by adding a DAV:response element for the search arbiter resource with 507 (Insufficient Storage) status.

>> Request:

```
SEARCH / HTTP/1.1
Host: example.net
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

 ... the query goes here ...
```

>> Response:

```
HTTP/1.1 207 Multistatus
Content-Type: text/xml; charset="utf-8"
Content-Length: 640

<?xml version="1.0" encoding="utf-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>http://www.example.net/sounds/unbrokenchain.au</D:href>
    <D:status>HTTP/1.1 200 OK</D:status>
  </D:response>
  <D:response>
    <D:href>http://tech.mit.example/arch96/photos/Lesh1.jpg</D:href>
    <D:status>HTTP/1.1 200 OK</D:status>
  </D:response>
  <D:response>
    <D:href>http://example.net</D:href>
    <D:status>HTTP/1.1 507 Insufficient Storage</D:status>
    <D:responsedescription xml:lang="en">
       Only first two matching records were returned
    </D:responsedescription>
  </D:response>
</D:multistatus>
```

2.4. Unsuccessful Responses

If a SEARCH request could not be executed or the attempt to execute it resulted in an error, the server MUST indicate the failure with an appropriate status code and SHOULD add a response body as defined in Section 1.6 of [RFC3253]. Unless otherwise stated, condition elements are empty; however, specific condition elements MAY include additional child elements that describe the error condition in more detail.

2.4.1. Example of an Invalid Scope

In the example below, a request failed because the scope identifies a HTTP resource that was not found.

>> Response:

```
HTTP/1.1 409 Conflict
Content-Type: text/xml; charset="utf-8"
Content-Length: 275

<?xml version="1.0" encoding="UTF-8"?>
<d:error xmlns:d="DAV:">
  <d:search-scope-valid>
    <d:response>
      <d:href>http://www.example.com/X</d:href>
      <d:status>HTTP/1.1 404 Object Not Found</d:status>
    </d:response>
  </d:search-scope-valid>
</d:error>
```

3. Discovery of Supported Query Grammars

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

Servers supporting the WebDAV extensions [RFC3253] and/or [RFC3744] MUST also:

- report SEARCH in the live property DAV:supported-method-set for all search arbiter resources, and
- support the live property DAV:supported-query-grammar-set as defined in Section 3.3.

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 as defined in Section 9.2 of [RFC2616].

If a resource supports the SEARCH method, then the server MUST list SEARCH in the Allow header defined in Section 14.7 of [RFC2616].

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" ":" 1#Coded-URL
Coded-URL  = <defined in Section 10.1 of [RFC4918]>
```

(This grammar uses the augmented BNF format defined in Section 2.1 of [RFC2616].)

The DASL response header indicates server support for query grammars in the OPTIONS method. The value is a list of URIs that indicate the types of supported grammars. Note that although the URIs can be used to identify each supported search grammar, there is not necessarily a direct relationship between the URI and the XML element name that can be used in XML based SEARCH requests (the element name itself is identified by its namespace name (a URI reference) and the element's local name).

> **Note:** this header field value is defined as a comma-separated list ([RFC2616], Section 4.2); thus, grammar URIs can appear in multiple header instances, separated by commas, or both.

For example:

```
DASL: <http://foobar.example/syntax1>,
      <http://akuma.example/syntax2>, <DAV:basicsearch>
DASL: <http://example.com/foo/natural-language-query>
```

3.3. DAV:supported-query-grammar-set (Protected)

This WebDAV property is required for any server supporting either [RFC3253] and/or [RFC3744] and identifies the XML-based query grammars that are supported by the search arbiter resource.

```
<!ELEMENT supported-query-grammar-set (supported-query-grammar*)>
<!ELEMENT supported-query-grammar (grammar)>
<!ELEMENT grammar ANY>
<!-- ANY value: a query grammar element type -->
```

3.4. Example: Grammar Discovery

This example shows that the server supports search on the /somefolder resource with the query grammars: DAV:basicsearch, http://foobar.example/syntax1 and http://akuma.example/syntax2. Note that servers supporting WebDAV SEARCH MUST support DAV:basicsearch.

>> Request:

```
OPTIONS /somefolder HTTP/1.1
Host: example.org
```

>> Response:

```
HTTP/1.1 200 OK
Allow: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, COPY, MOVE
Allow: MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH
DASL: <DAV:basicsearch>
DASL: <http://foobar.example/syntax1>, <http://akuma.example/syntax2>
```

This example shows the equivalent taking advantage of a server's support for DAV:supported-method-set and DAV:supported-query-grammar-set.

>> Request:

```
PROPFIND /somefolder HTTP/1.1
Host: example.org
Depth: 0
Content-Type: text/xml; charset="utf-8"
Content-Length: 165

<?xml version="1.0" encoding="UTF-8" ?>
<propfind xmlns="DAV:">
  <prop>
    <supported-query-grammar-set/>
    <supported-method-set/>
  </prop>
</propfind>
```

>> Response:

```
HTTP/1.1 207 Multi-Status
Content-Type: text/xml; charset="utf-8"
Content-Length: 1349

<?xml version="1.0" encoding="utf-8" ?>
<multistatus xmlns="DAV:">
 <response>
  <href>http://example.org/somefolder</href>
  <propstat>
   <prop>
    <supported-query-grammar-set>
     <supported-query-grammar>
      <grammar><basicsearch/></grammar>
     </supported-query-grammar>
     <supported-query-grammar>
      <grammar><syntax1 xmlns="http://foobar.example/"/></grammar>
     </supported-query-grammar>
     <supported-query-grammar>
      <grammar><syntax2 xmlns="http://akuma.example/"/></grammar>
     </supported-query-grammar>
    </supported-query-grammar-set>
    <supported-method-set>
     <supported-method name="COPY" />
     <supported-method name="DELETE" />
     <supported-method name="GET" />
     <supported-method name="HEAD" />
     <supported-method name="LOCK" />
     <supported-method name="MKCOL" />
     <supported-method name="MOVE" />
     <supported-method name="OPTIONS" />
     <supported-method name="POST" />
     <supported-method name="PROPFIND" />
     <supported-method name="PROPPATCH" />
     <supported-method name="PUT" />
     <supported-method name="SEARCH" />
     <supported-method name="TRACE" />
     <supported-method name="UNLOCK" />
    </supported-method-set>
   </prop>
   <status>HTTP/1.1 200 OK</status>
  </propstat>
 </response>
</multistatus>
```

Note that the query grammar element names marshalled as part of the DAV:supported-query-grammar-set can be directly used as element names in an XML-based query.

4. Query Schema Discovery: QSD

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

The DASL response header and the DAV:supported-query-grammar-set property provide 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 an 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 be 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.

4.1. Additional SEARCH Semantics

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 and with a root element of DAV:query-schema-discovery rather than DAV:searchrequest.

Marshalling:

The request body MUST be a DAV:query-schema-discovery element.

```
<!ELEMENT query-schema-discovery ANY>
<!-- ANY value: XML element specifying the query grammar
                and the scope -->
```

The response body takes the form of a DAV:multistatus element ([RFC4918], Section 13), where DAV:response is extended to hold the returned query grammar inside a DAV:query-schema container element.

```
<!ELEMENT response (href, status, query-schema?,
  responsedescription?) >
<!ELEMENT query-schema ANY>
```

The content of this container 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.

4.1.1. Example of Query Schema Discovery

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

>> Request:

```
SEARCH / HTTP/1.1
Host: recipes.example
Content-Type: application/xml; charset="utf-8"
Content-Length: 258

<?xml version="1.0"?>
<query-schema-discovery xmlns="DAV:">
  <basicsearch>
    <from>
      <scope>
        <href>http://recipes.example</href>
        <depth>infinity</depth>
      </scope>
    </from>
  </basicsearch>
</query-schema-discovery>
```

>> Response:

```
HTTP/1.1 207 Multistatus
Content-Type: application/xml; charset="utf-8"
Content-Length: xxx

<?xml version="1.0"?>
<multistatus xmlns="DAV:">
  <response>
    <href>http://recipes.example</href>
    <status>HTTP/1.1 200 OK</status>
    <query-schema>
      <basicsearchschema>
        <!-- (See Section 5.19 for
        the actual contents) -->
      </basicsearchschema>
    </query-schema>
  </response>
</multistatus>
```

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

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 other grammars.

DAV:basicsearch has several components:

- 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

<!-- "basicsearch" element -->

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

<!-- "select" element -->

```
<!ELEMENT select        (allprop | prop) >
```

<!-- "from" element -->

```
<!ELEMENT from          (scope+) >
<!ELEMENT scope         (href, depth, include-versions?) >
<!ELEMENT include-versions EMPTY >
```

```
<!-- "where" element -->

<!ENTITY % comp_ops     "eq | lt | gt| lte | gte">
<!ENTITY % log_ops      "and | or | not">
<!ENTITY % special_ops  "is-collection | is-defined |
                         language-defined | language-matches">
<!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|typed-literal)) >
<!ATTLIST lt            caseless   (yes|no) #IMPLIED>

<!ELEMENT lte           (prop, (literal|typed-literal)) >
<!ATTLIST lte           caseless   (yes|no) #IMPLIED>

<!ELEMENT gt            (prop, (literal|typed-literal)) >
<!ATTLIST gt            caseless   (yes|no) #IMPLIED>

<!ELEMENT gte           (prop, (literal|typed-literal)) >
<!ATTLIST gte           caseless   (yes|no) #IMPLIED>

<!ELEMENT eq            (prop, (literal|typed-literal)) >
<!ATTLIST eq            caseless   (yes|no) #IMPLIED>

<!ELEMENT literal       (#PCDATA)>
<!ELEMENT typed-literal (#PCDATA)>
<!ATTLIST typed-literal xsi:type CDATA #IMPLIED>

<!ELEMENT is-collection EMPTY >
<!ELEMENT is-defined    (prop) >

<!ELEMENT language-defined    (prop) >
<!ELEMENT language-matches    (prop, literal) >

<!ELEMENT like          (prop, literal) >
<!ATTLIST like          caseless   (yes|no) #IMPLIED>

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

```
<!-- "orderby" element -->

<!ELEMENT orderby       (order+) >
<!ELEMENT order         ((prop | score), (ascending | descending)?)>
<!ATTLIST order         caseless   (yes|no) #IMPLIED>
<!ELEMENT ascending     EMPTY>
<!ELEMENT descending    EMPTY>

<!-- "limit" element -->

<!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 sorted ascending by size.

```
<d:searchrequest xmlns:d="DAV:">
  <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, which is a set of properties and values. This document defines two possible values: DAV:allprop and DAV:prop, both defined in Section 14 of [RFC4918].

5.4. DAV:from

```
<!ELEMENT scope            (href, depth, include-versions?) >
<!ELEMENT include-versions EMPTY >
```

DAV:from defines the query scope. This contains one or more DAV:scope elements. Support for multiple scope elements is optional, however servers MUST fail a request specifying multiple DAV:scope elements if they can't support it (see Section 2.2.2, precondition DAV:search-multiple-scope-supported). The scope element contains mandatory DAV:href and DAV:depth elements.

DAV:href indicates the URI reference ([RFC3986], Section 4.1) to use as a scope.

When the scope is a collection, if DAV:depth is "0", the search includes only the collection. When it is "1", the search includes the collection and its immediate children. When it is "infinity", it includes the collection and all its progeny.

When the scope is not a collection, the depth is ignored and the search applies just to the resource itself.

If the server supports WebDAV Redirect Reference Resources ([RFC4437]) and the search scope contains a redirect reference resource, then it applies only to that resource, not to its target.

When the child element DAV:include-versions is present, the search scope will include all versions (see [RFC3253], Section 2.2.1) of all version-controlled resources in scope. Servers that do support versioning but do not support the DAV:include-versions feature MUST signal an error if it is used in a query (see Section 2.2.2, precondition DAV:search-scope-valid).

5.4.1. Relationship to the Request-URI

If the DAV:scope element is a URI ([RFC3986], Section 3), the scope is exactly that URI.

If the DAV:scope element is a relative reference ([RFC3986], Section 4.2), the scope is taken to be relative to the Request-URI.

5.4.2. Scope

A Scope can be an arbitrary URI reference.

Servers, of course, may support only particular scopes. This may include limitations for particular schemes such as "http:" or "ftp:" or certain URI namespaces. However, WebDAV-compliant search arbiters minimally SHOULD support scopes that match their own URI.

5.5. DAV:where

The 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 contains 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 PROPFIND for a property value would yield a non-2xx (see Section 10.2 of [RFC2616]) 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 than" a string with length greater than zero.

The DAV:is-defined operator is defined to test if the value of a property is not NULL.

5.5.4. Treatment of Properties with Mixed/Element Content

Comparisons of properties that do not have simple types (text-only content) is out of scope for the standard operators defined for DAV:basicsearch and therefore is defined to be UNKNOWN (as per Appendix A). For querying the DAV:resourcetype property, see Section 5.13.

5.5.5. Example: Testing for Equality

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

```
<d:where xmlns:d='DAV:'>
  <d:eq>
    <d:prop>
      <d:getcontentlength/>
    </d:prop>
    <d:literal>100</d:literal>
  </d:eq>
</d:where>
```

5.5.6. 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 of type "image/gif" over 4K in size.

```
<D:where xmlns:D='DAV:'>
  <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 (this is compatible with [SQL99]).

The "caseless" attribute may be used to indicate case-sensitivity for comparisons (Section 5.18).

5.6.1. Example of Sorting

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

```
<d:orderby xmlns:d='DAV:' xmlns:r='http://example.com/ns'>
  <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 "caseless" attribute may be used with this element (Section 5.18).

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, using less-than, less-than or equal, greater-than, and greater-than or equal, respectively. The "caseless" attribute may be used with these elements (Section 5.18).

5.10. DAV:literal

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

White space in literal values is significant in comparisons. For consistency with [RFC4918], clients SHOULD NOT specify the attribute "xml:space" (Section 2.10 of [XML]) to override this behavior.

In comparisons, the contents of DAV:literal SHOULD be treated as string, with the following exceptions:

- when operand for a comparison with a DAV:getcontentlength property, it SHOULD be treated as an unsigned integer value (the behavior for values not in this format is undefined),

- when operand for a comparison with a DAV:creationdate or DAV:getlastmodified property, it SHOULD be treated as a date value in the ISO-8601 subset defined for the DAV:creationdate property (see Section 15.1 of [RFC4918]; the behavior of values not in this format is undefined),
- when operand for a comparison with a property for which the type is known and when compatible with that type, it MAY be treated according to this type.

5.11. DAV:typed-literal (Optional)

There are situations in which a client may want to force a comparison not to be string-based (as defined for DAV:literal). In these cases, a typed comparison can be enforced by using DAV:typed-literal instead.

```
<!ELEMENT typed-literal (#PCDATA)>
```

The data type is specified using the xsi:type attribute defined in Section 2.6.1 of [XS1]. If the type is not specified, it defaults to "xs:string".

A server MUST reject a request using an unknown type with a status of 422 (Unprocessable Entity). It SHOULD reject a request if the value provided in DAV:typed-literal cannot be cast to the specified type.

The comparison evaluates to UNKNOWN if the property value cannot be cast to the specified datatype (see [XPATHFUNC], Section 17).

5.11.1. Example for Typed Numerical Comparison

Consider a set of resources with the dead property "edits" in the namespace "http://ns.example.org":

URI	property value
/a	"-1"
/b	"01"
/c	"3"
/d	"test"
/e	(undefined)

The expression

```
<lt xmlns="DAV:"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <prop><edits xmlns="http://ns.example.org"/></prop>
  <typed-literal xsi:type="xs:integer">3</typed-literal>
</lt>
```

will evaluate to TRUE for the resources "/a" and "/b" (their property values can be parsed as type xs:integer, and the numerical comparison evaluates to true), to FALSE for "/c" (property value is compatible, but numerical comparison evaluates to false), and UNKNOWN for "/d" and "/e" (the property either is undefined, or its value cannot be parsed as xs:integer).

5.12. Support for Matching xml:lang Attributes on Properties

The following two optional operators can be used to express conditions on the language of a property value (as expressed using the xml:lang attribute).

5.12.1. DAV:language-defined (Optional)

```
<!ELEMENT language-defined (prop)>
```

This operator evaluates to TRUE if the language for the value of the given property is known, FALSE if it isn't, and UNKNOWN if the property itself is not defined.

5.12.2. DAV:language-matches (Optional)

```
<!ELEMENT language-matches (prop, literal)>
```

This operator evaluates to TRUE if the language for the value of the given property is known and matches the language name given in the <literal> element, FALSE if it doesn't match, and UNKNOWN if the property itself is not defined.

Languages are considered to match if they are the same, or if the language of the property value is a sublanguage of the language specified in the <literal> element (see Section 4.3 of [XPATH], "lang function").

5.12.3. Example of Language-Aware Matching

The expression below will evaluate to TRUE if the property "foobar" exists and its language is either unknown, English, or a sublanguage of English.

```
<or xmlns="DAV:">
  <not>
    <language-defined>
      <prop><foobar/></prop>
    </language-defined>
  </not>
  <language-matches>
    <prop><foobar/></prop>
    <literal>en</literal>
  </language-matches>
</or>
```

5.13. DAV:is-collection

The DAV:is-collection operator allows clients to determine whether a resource is a collection (that is, whether its DAV:resourcetype element contains the element DAV:collection).

Rationale: This operator 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.

5.13.1. Example of DAV:is-collection

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

```
<where xmlns="DAV:">
  <is-collection/>
</where>
```

5.14. DAV:is-defined

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

Example:

```
<d:is-defined xmlns:d='DAV:' xmlns:x='http://example.com/ns'>
  <d:prop><x:someprop/></d:prop>
</d:is-defined>
```

5.15. 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.

5.15.1. Syntax for the Literal Pattern

```
pattern       = [wildcard] 0*( text [wildcard] )

wildcard      = exactlyone / zeroormore
text          = 1*( character / escapeseq )

exactlyone    = "_"
zeroormore    = "%"
escapechar    = "\"
escapeseq     = escapechar ( exactlyone / zeroormore / escapechar )

; character: see [XML], Section 2.2, minus wildcard / escapechar
character     = HTAB / LF / CR ; whitespace
character     =/ %x20-24 / %x26-5B / %x5D-5E / %x60-D7FF
character     =/ %xE000-FFFD / %x10000-10FFFF
```

(Note that the ABNF above is defined in terms of Unicode code points ([UNICODE5]); when a query is transmitted as an XML document over WebDAV, these characters are typically encoded in UTF-8 or UTF-16.)

The value for the literal is composed of wildcards separated by segments of text. Wildcards may begin or end the literal.

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.

5.15.2. 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 xmlns:D='DAV:'>
  <D:like caseless="yes">
    <D:prop><D:getcontenttype/></D:prop>
    <D:literal>image/%</D:literal>
  </D:like>
</D:where>
```

5.16. 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 the 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 at the discretion of the server implementation.

The following non-exhaustive list enumerates things that 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. Character canonicalization such as that done by the Unicode collation algorithm may or may not be applied.

5.16.1. Result Scoring (DAV:score Element)

Servers SHOULD indicate scores for the DAV:contains condition by adding a DAV:score XML element to the DAV:response element. Its value is defined only in the context of a particular query result. 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).

Modified DTD fragment for DAV:propstat:

```
<!ELEMENT response (href, ((href*, status)|(propstat+)),
                    responsedescription?, score?) >
<!ELEMENT score    (#PCDATA) >
```

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

5.16.2. Ordering by Score

To order search results by their score, the DAV:score element may be added as child to the DAV:orderby element (in place of a DAV:prop element).

5.16.3. Examples

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 xmlns:D='DAV:'>
  <D:contains>Peter Forsberg</D:contains>
</D:where>
```

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

```
<D:where xmlns:D='DAV:'>
  <D:and>
    <D:contains>Peter</D:contains>
    <D:contains>Forsberg</D:contains>
  </D:and>
</D:where>
```

5.17. Limiting the Result Set

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

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. The DAV:nresults XML element contains a requested maximum number of DAV:response elements to be returned in the response body. The server MAY disregard this limit. The value of this element is an unsigned integer.

5.17.1. Relationship to Result Ordering

If the result set is both limited by DAV:limit and ordered according to DAV:orderby, the results that are included in the response document SHOULD be those that order highest.

5.18. The 'caseless' XML Attribute

The "caseless" attribute allows clients to specify caseless matching behavior instead of character-by-character matching for DAV:basicsearch operators.

The possible values for "caseless" are "yes" or "no". The default value is server-specified. Caseless matching SHOULD be implemented as defined in Section 5.18[1] of the Unicode Standard ([UNICODE5]).

Support for the "caseless" attribute is optional. A server should respond with a status of 422 if it is used but cannot be supported.

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:

1. the set of properties that may be either searched, returned, or used to sort, and a hint about the data type of such properties.
2. the set of optional operators defined by the resource.

[1] http://www.unicode.org/versions/Unicode5.0.0/ch05.pdf#G21180

5.19.1. DTD for DAV:basicsearch QSD

```
<!ELEMENT basicsearchschema  (properties, operators)>
<!ELEMENT any-other-property EMPTY>
<!ELEMENT properties         (propdesc*)>
<!ELEMENT propdesc           ((prop|any-other-property), datatype?,
                              searchable?, selectable?, sortable?,
                              caseless?)>
<!ELEMENT operators          (opdesc*)>
<!ELEMENT opdesc             ANY>
<!ATTLIST opdesc             allow-pcdata (yes|no) #IMPLIED>
<!ELEMENT operand-literal    EMPTY>
<!ELEMENT operand-typed-literal EMPTY>
<!ELEMENT operand-property   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 five descriptions. The first, DAV:datatype, provides a hint about the type of the property value, and may be useful to a user interface prompting for a value. The remaining four (DAV:searchable, DAV:selectable, DAV:sortable, and DAV:caseless) identify portions of the query (DAV:where, DAV:select, and DAV:orderby, respectively). 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.

5.19.2.1. DAV:any-other-property

This element can be used in place of DAV:prop to describe properties of WebDAV properties not mentioned in any other DAV:prop element. For instance, this can be used to indicate that all other properties are searchable and selectable without giving details about their types (a typical scenario for dead properties).

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. Data types are identified by an element name. Where appropriate, a server SHOULD use the simple data types defined in [XS2].

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

Examples from [XS2], Section 3:

Qualified name	Example
xs:boolean	true, false, 1, 0
xs:string	Foobar
xs:dateTime	1994-11-05T08:15:5Z
xs:float	.314159265358979E+1
xs:integer	-259, 23

If the data type of a property is not given, then the data type defaults to xs: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:caseless Property Description

This element only applies to properties whose data type is "xs:string" and derived data types as per the DAV:datatype property description. Its presence indicates that comparisons performed for searches, and the comparisons for ordering results on the string property will be caseless (the default is character by character).

```
<!ELEMENT caseless 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, contained in an DAV:opdesc element, consists of the operator (as an empty element), followed by one element for each operand. The operand MUST be either DAV:operand-property, DAV:operand-literal, or DAV:operand-typed-literal, which indicate that the operand in the corresponding position is a property, a literal value, or a typed literal value, respectively. If an operator is polymorphic (allows more than one operand syntax) then each permitted syntax MUST be listed separately.

The DAV:opdesc element MAY have a "allow-pcdata" attribute (defaulting to "no"). A value of "yes" indicates that the operator can contain character data, as it is the case with DAV:contains (see Section 5.16). Definition of additional operators using this format is NOT RECOMMENDED.

```
<operators xmlns='DAV:'>
  <opdesc>
    <like/><operand-property/><operand-literal/>
  </opdesc>
</operators>
```

5.19.9. Example of Query Schema for DAV:basicsearch

```
<D:basicsearchschema xmlns:D="DAV:"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <D:properties>
    <D:propdesc>
      <D:prop><D:getcontentlength/></D:prop>
      <D:datatype><xs:nonNegativeInteger/></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><fstop xmlns="http://ns.example.org"/></D:prop>
      <D:selectable/>
    </D:propdesc>
    <D:propdesc>
      <D:any-other-property/>
      <D:searchable/><D:selectable/>
    </D:propdesc>
  </D:properties>
  <D:operators>
    <D:opdesc>
      <D:like/><D:operand-property/><D:operand-literal/>
    </D:opdesc>
    <D:opdesc allow-pcdata="yes">
      <D:contains/>
    </D:opdesc>
  </D:operators>
</D:basicsearchschema>
```

This response lists four properties. The data type of the last three properties is not given, so it defaults to xs: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:contains and DAV:like are supported.

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

6. Internationalization Considerations

Properties may be language-tagged using the xml:lang attribute (see [RFC4918], Section 4.3). The optional operators DAV:language-defined (Section 5.12.1) and DAV:language-matches (Section 5.12.2) allow the expression of conditions on the language tagging information.

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 ([RFC2616] and WebDAV ([RFC4918]) also apply to DASL. In addition, this section will include security risks inherent in the search and retrieval of resource properties and content.

A query MUST NOT allow clients to retrieve information that wouldn't have been available through the GET or PROPFIND methods in the first place. In particular:

- Query constraints on WebDAV properties for which the client does not have read access need to be evaluated as if the property did not exist (see Section 5.5.3).
- Query constraints on content (as with DAV:contains, defined in Section 5.16) for which the client does not have read access need to be evaluated as if a GET would return a 4xx status code.

A 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.

7.1. Implications of XML External Entities

XML supports a facility known as "external entities", defined in Section 4.2.2 of [XML], which instruct an XML processor to retrieve and perform an inline include of XML located at a particular URI. An external XML entity can be used to append or modify the document type declaration (DTD) associated with an XML document. An external XML entity can also be used to include XML within the content of an XML document. For non-validating XML, such as the XML used in this specification, including an external XML entity is not required by [XML]. However, [XML] does state that an XML processor may, at its discretion, include the external XML entity.

External XML entities have no inherent trustworthiness and are subject to all the attacks that are endemic to any HTTP GET request. Furthermore, it is possible for an external XML entity to modify the DTD, and hence affect the final form of an XML document, in the worst case significantly modifying its semantics, or exposing the XML processor to the security risks discussed in [RFC3023]. Therefore, implementers must be aware that external XML entities should be treated as untrustworthy.

There is also the scalability risk that would accompany a widely deployed application that made use of external XML entities. In this situation, it is possible that there would be significant numbers of requests for one external XML entity, potentially overloading any server that fields requests for the resource containing the external XML entity.

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. IANA Considerations

This document uses the namespace defined in Section 21 of [RFC4918] for XML elements.

9.1. HTTP Headers

This document specifies the HTTP header listed below, which has been added to the permanent HTTP header registry defined in [RFC3864].

9.1.1. DASL

Header field name:	DASL
Applicable protocol:	http
Status:	standard
Author/Change controller:	IETF
Specification document:	this specification (Section 3.2)

10. Contributors

This document is based on prior work on the DASL protocol done by the WebDAV DASL working group until the year 2000 -- namely by Alan Babich, Jim Davis, Rick Henderson, Dale Lowry, Saveen Reddy, Surendra Reddy, and Judith Slein (see <http://www.webdav.org/dasl/> for the working group's web site, <http://purl.org/NET/webdav/dasl-references/reqs> for a requirements document, and <http://purl.org/NET/webdav/dasl-references/dasl-protocol-00> for an early version of the specification).

11. Acknowledgements

This document has benefited from thoughtful discussion by Lisa Dusseault, Javier Godoy, Sung Kim, Chris Newman, Elias Sinderson, Martin Wallmer, Keith Wannamaker, Jim Whitehead, and Kevin Wiggen.

12. References

12.1. Normative References

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3253] Clemm, G., Amsden, J., Ellison, T., Kaler, C., and J. Whitehead, "Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning)", RFC 3253, March 2002.

[RFC3744] Clemm, G., Reschke, J., Sedlar, E., and J. Whitehead, "Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol", RFC 3744, May 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4918] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, June 2007.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C REC-xml-20060816, August 2006, <http://www.w3.org/TR/2006/REC-xml-20060816>.

[XPATH] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", W3C REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

[XPATHFUNC] Malhotra, A., Melton, J., and N. Walsh, "XQuery 1.0 and XPath 2.0 Functions and Operators", W3C REC-xpath-functions-20070123, January 2007, <http://www.w3.org/TR/2007/REC-xpath-functions-20070123/>.

[XS1] Thompson, H., Beech, D., Maloney, M., Mendelsohn, N., and World Wide Web Consortium, "XML Schema Part 1: Structures", W3C REC-xmlschema-1-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>.

[XS2] Biron, P., Malhotra, A., and World Wide Web Consortium, "XML Schema Part 2: Datatypes Second Edition", W3C REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>.

12.2. Informative References

[BCP47] Phillips, A. and M. Davis, "Matching of Language Tags", BCP 47, RFC 4647, September 2006.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

[RFC4437] Whitehead, J., Clemm, G., and J. Reschke, Ed., "Web Distributed Authoring and Versioning (WebDAV) Redirect Reference Resources", RFC 4437, March 2006.

[RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, March 2007.

[SQL99] Milton, J., "Database Language SQL Part 2: Foundation (SQL/Foundation)", ISO ISO/IEC 9075-2:1999 (E), July 1999.

[UNICODE5] The Unicode Consortium, "The Unicode Standard - Version 5.0", Addison-Wesley, November 2006, <http://www.unicode.org/versions/Unicode5.0.0/>.
ISBN 0321480910[2]

[WEBDAV-BIND] Clemm, G., Crawford, J., Reschke, J., Ed., and J. Whitehead, "Binding Extensions to Web Distributed Authoring and Versioning (WebDAV)", October 2008.

[2] urn:isbn:0321480910

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, 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:

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

Appendix B. Candidates for Future Protocol Extensions

This section summarizes issues that have been raised during the development of this specification, but for which no resolution could be found with the constraints in place. Future revisions of this specification should revisit these issues, though.

B.1. Collation Support

Matching and sorting of textual data relies on collations. With respect to WebDAV SEARCH, a combination of various design approaches could be used:

- Require server support for specific collations.
- Require that the server can advertise which collations it supports.
- Allow a client to select the collation to be used.

In practice, the current implementations of WebDAV SEARCH usually rely on backends they do not control, and for which collation information may not be available. To make things worse, implementations of the DAV:basicsearch grammar frequently need to combine data from multiple underlying stores (such as properties and full text content), and thus collation support may vary based on the operator or property.

Another open issue is what collation formalism to support. At the time of this writing, the two specifications below seem to provide the necessary framework and thus may be the base for future work on collation support in WebDAV SEARCH:

1. "Internet Application Protocol Collation Registry" ([RFC4790]).
2. "XQuery 1.0 and XPath 2.0 Functions and Operators" ([XPATHFUNC], Section 7.3.1).

B.2. Count

DAV:basicsearch does not allow a request that returns the count of matching resources.

A protocol extension would need to extend DAV:select, and also modify the DAV:multistatus response format.

B.3. Diagnostics for Unsupported Queries

There are many reasons why a given query may not be supported by a server. Query Schema Discovery (Section 4) can be used to discover some constraints, but not all.

Future revisions should consider the introduction of specific condition codes ([RFC4918], Section 16) to these situations.

B.4. Language Matching

Section 5.12.2 defines language matching in terms of the XPath "lang" function ([XPATH], Section 4.3). Future revisions should consider building on [BCP47] instead.

B.5. Matching Media Types

Matching media types using the DAV:getcontenttype property and the DAV:like operator is hard due to DAV:getcontenttype also allowing parameters. A new operator specifically designed for the purpose of matching media types probably would simplify things a lot. See <http://lists.w3.org/Archives/Public/www-webdav-dasl/2003OctDec/0109.html> for a specific proposal.

B.6. Query by Name

DAV:basicsearch operates on the properties (and optionally the contents) of resources, and thus doesn't really allow matching on parts of the resource's URI. See <http://lists.w3.org/Archives/Public/www-webdav-dasl/2003OctDec/0100.html> for a proposed extension covering this use case.

B.7. Result Paging

A frequently discussed feature is the ability to specifically request the "next" set of results, when either the server decided to truncate the result, or the client explicitly asked for a limited set (for instance, using the DAV:limit element defined in Section 5.17).

In this case, it would be desirable if the server could keep the full query result, and provide a new URI identifying a separate result resource, allowing the client to retrieve additional data through GET requests, and remove the result through a DELETE request.

B.8. Search Scope Discovery

Given a Search Arbiter resource, there's currently no way to discover programmatically the supported sets of search scopes. Future revisions of this specification could specify a scope discovery mechanism, similar to the Query Schema Discovery defined in Section 4.

Index

Authors' Addresses

Julian F. Reschke (editor)
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany
Phone: +49 251 2807760
EMail: julian.reschke@greenbytes.de
URI: http://greenbytes.de/tech/webdav/

Surendra Reddy
Mitrix, Inc.
303 Twin Dolphin Drive, Suite 600-37
Redwood City, CA 94065
U.S.A.
Phone: +1 408 500 1135
EMail: Surendra.Reddy@mitrix.com

Jim Davis
27 Borden Street
Toronto, Ontario M5S 2M8
Canada
Phone: +1 416 929 5854
EMail: jrd3@alum.mit.edu
URI: http://www.econetwork.net/~jdavis

Alan Babich
IBM Corporation
3565 Harbor Blvd.
Costa Mesa, CA 92626
U.S.A.
Phone: +1 714 327 3403
EMail: ababich@us.ibm.com