Network Working Group
Internet-Draft
Expires: August 6, 2004

J. Reschke, Ed.
greenbytes
S. Reddy
Oracle
J. Davis
Intelligent Markets
A. Babich
Filenet

February 6, 2004

WebDAV SEARCH
draft-reschke-webdav-search-06

Status of this Memo

Copyright Notice

Abstract

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

Distribution of this document is unlimited. Please send comments to the Distributed Authoring and Versioning (WebDAV) DASL mailing list

at www-webdav-dasl@w3.org [1], which may be joined by sending a message with subject "subscribe" to www-webdav-dasl-request@w3.org [2]. Discussions of the WebDAV DASL mailing list are archived at URL: http://lists.w3.org/Archives/Public/www-webdav-dasl/.

Table of Contents

1. Introduction

1.1 DASL

This document defines WebDAV SEARCH, 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. It is based on the expired draft for WebDAV DASL [DASL]. [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:query-schema-discovery XML element,
- the DAV:basicsearch XML element and query grammar, and
- the DAV:basicsearchschema XML element.

For WebDAV-compliant servers, it also defines a new live property DAV:supported-query-grammar-set.

1.2 Relationship to DAV

DASL relies on the resource and property model defined by [RFC2518]. 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], in [RFC2518], in [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 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

The augmented BNF used by this document to describe protocol elements is exactly the same as the one described in Section 2.1 of [RFC2616]. Because this augmented BNF uses the basic production rules provided in Section 2.2 of [RFC2616], 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].

This document uses XML DTD fragments as a purely notational convention. WebDAV request and response bodies can not be validated due to the specific extensibility rules defined in section 23 of [RFC2518] 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.

Similarily, 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.

1.5 Editorial note on usage of 'DAV:' namespace

*Note that this draft currently defines elements and properties in the WebDAV namespace "DAV:" which it shouldn't do as it isn't a work item of the WebDAV working group. The reason for this is the desire

for some kind of backward compatibility to the expired DASL drafts and the assumption that the draft may become an official RFC submission of the WebDAV working group at a later point of time.*

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 [RFC2518] 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 an entity matching the [RFC2518] 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. 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 [RFC2518]), 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. It's 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 [RFC2518] 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 match that of a PROPFIND. 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 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 all of these URIs. Clients can use the live property DAV:resource-id defined in [BIND] to identify possible duplicates.

2.3.1 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.

2.3.2 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 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
Content-Length: xxx

<?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: xxx

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

2.3.3 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 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.

>> 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: xxx

<?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:propstat>
      <D:prop/>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
  <D:response>
    <D:href>http://tech.mit.test/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://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 [RFC3253], section 1.6. Unless otherwise stated, condition elements are empty, however specific conditions element 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: xxx

<?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 [ACL] 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 defined in [RFC2616].

If a resource supports the SEARCH method, then the server MUST list SEARCH in the OPTIONS response as defined by [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" ":" Coded-URL-List
Coded-URL-List : Coded-URL [ "," Coded-URL-List ]
Coded-URL ; defined in section 9.4 of [RFC2518]
```

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. Note that although the URI 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 it's namespace name (a URI reference) and the element's local name).

This header MAY be repeated.

For example:

```
DASL: <http://foobar.test/syntax1>
DASL: <http://akuma.test/syntax2>
DASL: <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 [ACL] 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.test/syntax1 and http://akuma.test/syntax2. Note that every server 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.test/syntax1>
DASL: <http://akuma.test/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: xxx

<?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: xxx

<?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.test" /></grammar>
     </supported-query-grammar>
     <supported-query-grammar>
      <grammar><syntax2 xmlns="http://akuma.test/"/></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 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.

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 DAV:query-schema-discovery element.

```
<!ELEMENT query-schema-discovery ANY>
ANY value: XML element defining a valid query
```

The response body takes the form of a RFC2518 DAV:multistatus element, where DAV:response is extended to hold the returned query grammar inside a DAV:query-schema container element.

```
<!ELEMENT response (href, ((href*, status)|(propstat+)),
  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.test, the grammar is DAV:basicsearch, the scope is also recipes.test.

>> Request:

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

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

```
>> Response:

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

<?xml version="1.0"?>
<multistatus xmlns="DAV:">
  <response>
    <href>http://recipes.test</href>
    <status>HTTP/1.1 200 OK</status>
    <query-schema>
      <basicsearchschema>
        <!-- (See section "Query schema for DAV:basicsearch" 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) >

<!-- "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) >

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

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

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

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

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

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

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

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

   <!ELEMENT contains      (#PCDATA)>

   <!-- "orderby" element -->

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

```
<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 [RFC2518] and revised in [RFC3253].

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 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 (toplevel) members of the collection. When it is "infinity", the search includes all recursive members of the collection.

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

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.

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 is an absolute URI reference, 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.

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 [RFC2616], section 10.2) 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 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>
  <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 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 (this is compatible with [SQL99]).

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 "caseless" 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 for each author, 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 "caseless" 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, using less-than, less-than or equal, greater-than, and greater-than or equal respectively. The "caseless" attribute may be used with these elements.

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 [RFC2518], clients SHOULD NOT specify the attribute "xml:space" (section 2.10 of [XML]) to override this behaviour.

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 integer value (the behaviour for non-integer values 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 ([RFC2518], section 13.1).

- when operand for a comparison with a property for which the type is known, 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 [XS1], section 2.6.1. If the type is not specified, it defaults to "xs:string".

A server MUST reject a request with an unknown type.

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:number, 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 can not be parsed as xs:number).

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 [XPATH], section 4.3, "lang function").

5.12.3 Example of language-aware matching

The expression below will evaluate to TRUE if the property "foobar" exists and it's 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 it's 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>
  <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> | escapesequence )
exactlyone : = "_"
zeroormore := "%"
escapechar := "\"
escapesequence := "\" ( exactlyone | zeroormore | escapechar )
character: valid XML characters (see section 2.2 of [XML]),
           minus ( exactlyone | zeroormore | escapechar )
```

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>
  <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 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.

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. It's 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>
  <D:contains>Peter Forsberg</D:contains>
</D:where>
```

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>
```

5.17 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 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 must be those that order highest.

5.18 The 'caseless' XML attribute

The "caseless" attribute allows clients to specify caseless matching behaviour 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 of the Unicode Standard ([UNICODE4]).

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.

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>
<!ELEMENT operand-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. Datatypes are identified by an element name. Where appropriate, a server SHOULD use the simple datatypes 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 compares 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 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.

```
<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://jennicam.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:operators>
</D:basicsearchschema>
```

This response lists four properties. The datatype 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:is-defined 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 [RFC2518], section 4.4). The optional operators DAV:language-defined (Section 5.12.1) and DAV:language-matches (Section 5.12.2) allow to express 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 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.)

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 which 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 which 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. Authentication

Authentication mechanisms defined in WebDAV will also apply to DASL.

10. IANA Considerations

This document uses the namespace defined by [RFC2518] for XML elements. All other IANA considerations mentioned in [RFC2518] are also applicable to DASL.

11. 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 and Surendra Reddy.

12. Acknowledgements

This document has benefited from thoughtful discussion by Lisa Dusseault, Sung Kim, Elias Sinderson, Martin Wallmer, Jim Whitehead and Kevin Wiggen.

Normative References

[ACL] Clemm, G., Hopkins, A., Sedlar, E. and J. Whitehead, "WebDAV Access Control Protocol", ID draft-ietf-webdav-acl-13, January 2004, <http://www.webdav.org/acl/protocol/draft-ietf-webdav-acl-13.htm>.

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

[RFC2518] Goland, Y., Whitehead, E., Faizi, A., Carter, S. and D. Jensen, "HTTP Extensions for Distributed Authoring -- WEBDAV", RFC 2518, February 1999.

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

[RFC3023] Makoto, 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", RFC 3253, March 2002.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E. and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C REC-xml-20040204, February 2004, <http://www.w3.org/TR/2004/REC-xml-20040204>.

[XMLNS] Bray, T., Hollander, D. and A. Layman, "Namespaces in XML", W3C REC-xml-names-19990114, January 1999, <http://www.w3.org/TR/1999/REC-xml-names-19990114>.

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

[XS1] Thompson, H., Beech, D., Maloney, M., Mendelsohn, N. and World Wide Web Consortium, "XML Schema Part 1: Structures", W3C REC-xmlschema-1-20010502, May 2001, <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>.

[XS2] Biron, P., Malhotra, A. and World Wide Web Consortium, "XML Schema Part 2: Datatypes", W3C REC-xmlschema-2-20010502, May 2001, <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>.

Informative References

[BIND] Clemm, G., Crawford, J., Reschke, J., Slein, J. and J. Whitehead, "Binding Extensions to WebDAV", ID draft-ietf-webdav-bind-03, December 2003, <http://greenbytes.de/tech/webdav/draft-ietf-webdav-bind-03.html>.

[DASL] Reddy, S., Lowry, D., Reddy, S., Henderson, R., Davis, J. and A. Babich, "DAV Searching & Locating", ID draft-dasl-protocol-00, July 1999, <http://www.webdav.org/dasl/protocol/draft-dasl-protocol-00.html>.

[DASLREQ] Davis, J., Reddy, S. and J. Slein, "Requirements for DAV Searching and Locating", ID draft-dasl-requirements-01, February 1999, <http://www.webdav.org/dasl/requirements/draft-dasl-requirements-01.html>.

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

[UNICODE4] The Unicode Consortium, "The Unicode Standard - Version 4.0", Addison-Wesley , August 2003, <http://www.unicode.org/versions/Unicode4.0.0/>.

ISBN 0321185781 [3]

URIs

[1] <mailto:www-webdav-dasl@w3.org>

[2] <mailto:www-webdav-dasl-request@w3.org?subject=subscribe>

[3] <urn:isbn:0321185781>

Authors' Addresses

Julian F. Reschke (editor)
greenbytes GmbH
Salzmannstrasse 152
Muenster, NW 48159
Germany

Phone: +49 251 2807760
Fax: +49 251 2807761
EMail: julian.reschke@greenbytes.de
URI: http://greenbytes.de/tech/webdav/

Surendra Reddy
Oracle Corporation
600 Oracle Parkway, M/S 6op3
Redwoodshores, CA 94065

Phone: +1 650 506 5441
EMail: Surendra.Reddy@oracle.com

Jim Davis
Intelligent Markets
410 Jessie Street 6th floor
San Francisco, CA 94103

EMail: jrd3@alum.mit.edu

Alan Babich
FileNET Corp.
3565 Harbor Blvd.
Costa Mesa, CA 92626

Phone: +1 714 327 3403
EMail: ababich@filenet.com

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:

UNKNOWN and UNKNOWN = UNKNOWN

UNKNOWN or UNKNOWN = 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
```

Appendix B. Change Log (to be removed by RFC Editor before publication)

B.1 From draft-davis-dasl-protocol-xxx

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

June 3, 1999 Cosmetic and minor editorial changes only. Fix nits
reported by Jim Whitehead in email of April 26, 1999. Converted to
HTML from Word 97, manually.

April 20, 2000 Removed redirection feature, since 301/302 suffices.
Removed Query Schema Discovery (former chapter 4). Everyone agrees
this is a useful feature, but it is apparently too difficult to
define at this time, and it is not essential for DASL.

B.2 since start of draft-reschke-webdav-search

October 09, 2001 Added Julian Reschke as author.
Chapter about QSD re-added.
Formatted into RFC2629-compliant XML document.
Added first comments.
ID version number kicked up to draft-dasl-protocol-03.

October 17, 2001 Updated address information for Jim Davis.
Added issue of datatype vocabularies.
Updated issue descriptions for grammar discovery, added issues on
query schema DTD.

Fixed typos in XML examples.

December 17, 2001 Re-introduced split between normative and non-normative references.

January 05, 2002 Version bumbed up to 04. Started work on resolving the issues identified in the previous version.

January 14, 2002 Fixed some XML typos.

January 22, 2002 Closed issues naming-of-elements. Fixed query search DTD and added option to discover properties of "other" (non-listed) properties.

January 25, 2002 Changed into private submission and added reference to historic DASL draft. Marked reference to DASL requirements non-normative.
Updated reference to latest deltav spec.

January 29, 2002 Added feedback from and updated contact info for Alan Babich.
Included open issues collected in http://www.webdav.org/dasl/protocol/issues.html.

February 8, 2002 Made sure that all artwork fits into 72 characters wide text.

February 18, 2002 Changed Insufficient storage handling (multistatus). Moved is-collection to operators and added to DTD. Made scope/depth mandatory.

February 20, 2002 Updated reference to SQL99.

February 28, 2002 "Non-normative References" -> "Informative References". Abstract updated. Consistently specify a charset when using text/xml (no change bars). Do not attempt to define PROPFIND's entity encoding (take out specific references to text/xml). Remove irrelevant headers (Connection:) from examples (no change bars). Added issue on querying based on DAV:href. Updated introduction to indicate relationship to DASL draft. Updated HTTP reference from RFC2068 to RFC2616. Updated XML reference to XML 1.0 2nd edition.

March 1, 2002 Removed superfluous namespace decl in 2.4.2. Reopened JW14 and suggest to drop xml:space support.

March 3, 2002 Removed "xml:space" feature on DAV:literal. Added issue about string comparison vs. collations vs. xml:lang. Updated some of the open issues with details from JimW's original mail in April 1999. Resolved scope vs relative URI references. Resolved issues about DAV:ascending (added to index) and the BNF for DAV:like (changed "octets" to "characters").

March 8, 2002 Updated reference to DeltaV (now RFC3253). Added Martin Wallmer's comments, moved JW5 into DAV:basicsearch section.

March 11, 2002 Closed open issues regaring the type of search arbiters (JW3) and their discovery (JW9). Rephrased requirements on multistatus response bodies (propstat only if properties were selected, removed requirement for responsedescription).

March 23, 2002 RFC2376 -> RFC3023. Added missing first names of authors. OPTIONS added to example for DAV:supported-method-set.

B.3 since draft-reschke-webdav-search-00

March 29, 2002 Abstract doesn't refer to DASL WG anymore.

April 7, 2002 Fixed section title (wrong property name supported-search-grammar-set. Changed DAV:casesensitve to "casesensitive" (it wasn't in the DAV: namespace after all).

May 28, 2002 Updated some issues with Jim Davis's comments.

June 10, 2002 Added proposal for different method for query schema discovery, not using pseudo-properties.

June 25, 2002 QSD marshalling rewritten. Added issue "isdefined-optional".

B.4 since draft-reschke-webdav-search-01

July 04, 2002 Added issue "scope-collection".

July 08, 2002 Closed issue "scope-collection".

August 12, 2002 Added issues "results-vs-binds" and "select-allprop".

October 22, 2002 Added issue "undefined-expressions".

November 18, 2002 Changed example host names (no change tracking).

November 25, 2002 Updated issue "DB2/DB7". Closed issues "undefined expressions", "isdefined-optional" and "select-allprop".

B.5 since draft-reschke-webdav-search-02

November 27, 2002 Added issues "undefined-properties", "like-exactlyone" and "like-wildcard-adjacent". Closed issue "query-on-href". Added acknowledgments section.

November 28, 2002 Closed issue "like-exactlyone". Added issue "mixed-content-properties".

December 14, 2002 Closed issues "undefined-properties", "results-vs-binds", "mixed-content-properties". Updated issue "like-wildcard-adjacent". Added informative reference to BIND draft. Updated reference to ACL draft.

January 9, 2003 Removed duplicate section on invalid scopes. Added comments to some open issues. Closed issues JW25/26, score-pseudo-property and null-ordering.

January 10, 2003 Issue limit-vs-ordering plus resolution. Closed issue JW17/JW24b.

January 14, 2003 New issue order-precedence. Started resolution of DB2/DB7.

January 15, 2003 Started spec of DAV:typed-literal.

January 17, 2003 Fix one DAV:like/DAV:getcontenttype example (add / to like expression, make case-insensitive).

January 28, 2003 Update issue(s) result-truncation, JW24d. Fixed response headers in OPTIONS example. Added issue qsd-optional. Closed issue(s) order-precedence, case-insensitivity-name.

February 07, 2003 Added issue scope-vs-versions. score-pseudo-property: allow DAV:orderby to explicitly specify DAV:score.

B.6 since draft-reschke-webdav-search-03

April 24, 2003 Fixed two "?" vs "_" issues (not updated in last draft).

June 13, 2003 Improve index.

B.7 since draft-reschke-webdav-search-04

July 7, 2003 Typo fixed (propstat without status element).

August 11, 2003 Remove superfluous IP and copyright sections.

September 09, 2003 Added issues "2.4-multiple-uris" and "5.1-name-filtering".

October 06, 2003 Fix misplaced section end in 5.11, add table formatting. Enhance table formatting in 5.18.3. Updated ACL and BIND references. Added XPATH reference. Closed issue JW24d by adding new optional operators. Updated more open issues, added issues from January meeting. Add K. Wiggen to Acknowledgements. Add Contributors section for the authors of the original draft. Close issue "scope-vs-versions" (optional feature added). Close (new) issue "1.3-import-DTD-terminology". Add issue "1.3-import-requirements-terminology".

October 07, 2003 Typos fixed. Moved statement about DAV: namespace usage into separate (sub-)section. Closed "1.3-import-requirements-terminology". Update I18N Considerations with new xml:lang support info (see issue JW24d). Close issue "DB2/DB7" (remaining typing issues are now summarized in issue "typed-literal"). Fix misplaced section end in section 7. Started change to use RFC3253-style method definitions and error marshalling.

October 08, 2003 Remove obsolete language that allowed reporting invalid scopes and such inside multistatus. Add new issue "5.4.2-scope-vs-redirects".

B.8 since draft-reschke-webdav-search-05

October 11, 2003 Separate DAV:basicsearch DTD into separate figures for better maintainability. Update DTD with language-* operators and typed-literal element (optional).

October 14, 2003 Close issue "5.4.2-multiple-scope".

November 04, 2003 Update reference from CaseMap to UNICODE4, section 5.18.

November 16, 2003 Updated issue "5.1-name-filtering".

November 24, 2003 Reformatted scope description (collection vs. non-collection).

November 30, 2003 Add issue "5_media_type_match".

February 6, 2004 Updated all references.

Appendix C. Resolved issues (to be removed by RFC Editor before publication)

Issues that were either rejected or resolved in this version of this document.

C.1 5.4.2-multiple-scope

Type: change

<http://lists.w3.org/Archives/Public/www-webdav-dasl/2003JulSep/0012.html>

prakash.yamuna@covigna.com (2003-09-27): (asks for the ability to specify multiple scopes in a single query)

julian.reschke@greenbytes.de (2003-10-03): Consider making this an optional extension iff we can come up with a simple enough definition of it's impact on sorting/ranking and so on. Otherwise propose to reject.

Resolution (2003-10-14): Allow servers to support multiple scopes in DAV:basicsearch. Make clear that this is optional. Define precondition accordingly.

Appendix D. Open issues (to be removed by RFC Editor prior to publication)

D.1 1.3-apply-condition-code-terminology

Type: change

julian.reschke@greenbytes.de (2003-10-07): (Umbrella issue that will be left open until RFC3253 condition terminlogy is used throughout the document)

D.2 2.4-multiple-uris

Type: change

julian.reschke@greenbytes.de (2003-09-09): However, the set of URIs for a given resource may be unlimited due to possible bind loops. Therefore consider to report just one URI per resource.

D.3 result-truncation

Type: change

<http://lists.w3.org/Archives/Public/www-webdav-dasl/2002JanMar/0163.html>

ldusseault@xythos.com (2002-03-29): I believe the same response body that contains the first N <DAV:response> elements should also contain a *different* element stating that the results were incomplete and the result set was truncated by the server. There may also be a need to report that the results were incomplete and the result set was truncated at the choice of the client (isn't there a limit set in the client request?) That's important so the client knows the difference between receiving 10 results because there were >10 but only 10 were asked for, and receiving 10 results because there were only exactly 10 results and it just happens that 10 were asked for.

jrd3@alum.mit.edu (2002-05-28): I agree that this could be useful, but I think this issue should be consolidated with issue JW5 (see below), which proposes that DASL basicsearch ought to have a way for client to request additional result sets. It should be moved because there is little or no value in allowing a client to distinguish between the case where "N results were requested, and there are exactly N available" and "N results were requested, and there are more than N available" if there is no way for client to get the next batch of results.

julian.reschke@greenbytes.de (2003-01-28): Feedback from interim WG

meeting: agreement that marshalling should be rewritten and backwards compatibility is not important. Proposal: extend DAV:multistatus by a new child element that indicates (1) the range that was returned, (2) the total number of results and (3) a URI identifying the result (for resubmission when getting the "next" results). Such as <multistatus xmlns='DAV:'> <search-result> <href>...identifier for result set...</href> <total><-- number of results --></total> <start><-- 1-based index of 1st result --></start> <length><-- size of result set returned --></length> <partial-result/><-- indicates that this is a partial result --> </search-result> ...response elements for search results... </multistatus> The example below would then translate to: HTTP/1.1 207 Multistatus Content-Type: text/xml; charset="utf-8" Content-Length: xxx <?xml version="1.0" encoding="utf-8"?> <D:multistatus xmlns:D="DAV:"> <D:search-result> <D:partial-result/> </D:search-result> <D:response> <D:href>http://www.example.net/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.test/archive96/photos/Lesh1.jpg</D:href> <D:propstat> <D:prop/> <D:status>HTTP/1.1 200 OK</D:status> </D:propstat> </D:response> </D:multistatus> Q: do we need all elements, in particular start and length?

julian.reschke@greenbytes.de (2003-10-07): Related: if this is supposed to be normative to DAV:basicsearch, it can't stay in an "example" sub-section.

D.4 qsd-optional

Type: change

julian.reschke@greenbytes.de (2003-01-28): WG January meeting feedback: QSD should be made required.

kwiggen@xythos.com (2003-10-03): (significant pushback, see mailing list thread at http://lists.w3.org/Archives/Public/www-webdav-dasl/2003OctDec/0003.html).

D.5 5.1-name-filtering

Type: change

<http://lists.w3.org/Archives/Public/www-webdav-dasl/2003JulSep/0002.html>

julian.reschke@greenbytes.de (2003-09-08): This query grammar supports properties and content, but not conditions on URL elements (such as the last segment that many WebDAV implementations treat as "file name"). Discuss possible extension such as adding name filters

to the scope, or adding a specific operator.

Martin.Wallmer@softwareag.com (2003-11-11): Specific proposal to add this feature as scope restriction, see <http://lists.w3.org/Archives/Public/www-webdav-dasl/2003OctDec/0035.html>.

Martin.Wallmer@softwareag.com (2003-11-25): Updated proposal: <http://lists.w3.org/Archives/Public/www-webdav-dasl/2003OctDec/0100.html>.

D.6 5_media_type_match

Type: change

julian.reschke@greenbytes.de (2003-11-30): Putting conditions on DAV:getcontenttype is hard (see <http://lists.w3.org/Archives/Public/www-webdav-dasl/2003OctDec/0101.html> is (too?) hard. Proposal for a specific operator for expressing conditions on the media type: <http://lists.w3.org/Archives/Public/www-webdav-dasl/2003OctDec/0109.html>.

D.7 5.4.2-scope-vs-redirects

Type: change

<http://lists.w3.org/Archives/Public/www-webdav-dasl/2003OctDec/0010.html>

julian.reschke@greenbytes.de (2003-10-08): Clarify the relation of scope and redirect (3xx) resources.

D.8 language-comparison

Type: change

<http://lists.w3.org/Archives/Public/www-webdav-dasl/2002JanMar/0122.html>

julian.reschke@greenbytes.de (2002-03-03): XPath/XQuery (see draft, and open issue) specify string comparisons based on collations, not languages. I think we should adopt this. This would mean that "xml:lang" would be removed, and an optional attribute specifying the name of the collation is added.

julian.reschke@greenbytes.de (2003-01-09): Proposal: adopt "lang" and "collation" attribute from XSLT 2.0's xsl:sort.

D.9 JW16b/JW24a

Type: change

<http://lists.w3.org/Archives/Public/www-webdav-dasl/1999AprJun/0002.html>

ejw@ics.uci.edu (2000-04-20): Define how comparisons on strings work, esp for i18n. Need policy statement about sort order in various national languages. (JW said "non-Latin" but it's an issue even in languages that use the latin char set.)

julian.reschke@greenbytes.de (2003-01-28): This issue not only applies to the comparison operators, but also to ordering!

D.10 typed-literal

Type: change

julian.reschke@greenbytes.de (2003-01-15): 1. (insert language defining the comparison following the rules defined in http://www.w3.org/TR/xpath20/#id-comparisons). 2. Extend Basicsearch QSD grammar to support discovery of typed-literal 3. Update DTD. 4. Discuss behaviour of DAV:literal when the property's type is known for the complete search scope (is the server allowed to be "smart"?)

julian.reschke@greenbytes.de (2003-10-11): 3.: done

Index

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.