Network Working Group
Internet-Draft
Expires: August 26, 2008

C. Daboo
Apple

February 23, 2008

vCard Extensions to WebDAV (CardDAV)
draft-daboo-carddav-04

Status of this Memo

Copyright Notice

Abstract

This document defines extensions to the Web Distributed Authoring and Versioning (WebDAV) protocol to specify a standard way of accessing, managing, and sharing contact information based on the vCard format.

Discussion of this Internet-Draft is taking place on the mailing list <http://lists.osafoundation.org/mailman/listinfo/ietf-carddav>.

Table of Contents

1. Introduction and Overview

Address books containing contact information are a key component of personal information management tools, such as email, calendaring and scheduling, and instant messaging clients. To date several protocols have been used for remote access to contact data, including Lightweight Directory Access Protocol LDAP [RFC2251], Internet Message Support Protocol IMSP [IMSP] and Application Configuration Access Protocol ACAP [RFC2244], together with SyncML used for synchronization of such data.

1.1. IMSP

IMSP [IMSP], which was the predecessor to ACAP [RFC2244], received limited support from vendors, but those that did implement solutions based on it, found it to be a useful feature for large deployments of email clients at sites where users may roam from machine to machine. IMSP provided for multiple personal, shared or public address books, organized in a hierarchy, and gave individual users the ability to control access to their address books so that they could grant read or write access rights to other specific users or groups. This provided an easy and convenient way for users or workgroups to quickly setup and manage shared address information. Address book support in IMSP suffers from a number of problems, including a limited format for the address data itself, and scalability issues with large address books.

The key features of address book support in IMSP are:

1. Ability to use multiple address books with hierarchical layout.

2. Ability to control access to individual address books.

3. Server-side searching of address data, avoiding the need for clients to download an entire address book in order to do a quick address 'expansion' operation.

4. Ability to download/upload an individual address in an address book.

The key disadvantages of address book support in IMSP are:

1. Limited schema for address data.

2. Does not scale to large address books (e.g. no way to page through the list of addresses in an address book).

3. Does not provide any type of synchronization capability, which easily leads to 'lost update' problems when multiple users are editing the same address book entries.

4. Lack of internationalization support.

5. Does not provide per-address access control.

6. Does not provide a simple way to lookup users on the system.

1.2. ACAP

ACAP [RFC2244] was meant as the successor to IMSP and as such was designed to be a more 'generic' data access protocol for general application use. ACAP defined specific 'datasets' (basically formal schema definitions) for different anticipated areas of use, including address books, email accounts, application preferences, mime types etc. The use of such formal schema definitions was intended to enhance interoperability between clients. However, ACAP proved difficult to implement due to over complexity in the protocol itself, and this lead to few implementations.

The key features of address book support in ACAP are:

1. Ability to use multiple address books with hierarchical layout.

2. Ability to control access to individual address books and address entries.

3. Server-side searching of address data, avoiding the need for clients to download an entire address book in order to do a quick address 'expansion' operation.

4. Ability to inherit address book data from others.

5. Ability to watch changes in address book data through use of 'contexts'.

6. Ability to page through address book data through use of 'contexts'.

7. Internationalization support through use of UTF-8 for all data.

8. Well defined address schema to enhance client interoperability.

9. Compatibility with vCard data format.

10. Users and groups dataset can be used to enumerate and find other users on the system.

The key disadvantages of address book support in ACAP are:

1. Inheritance, access control and contexts all together is hard, and ultimately proved one of the major hurdles to implementations.

1.3. LDAP

LDAP [RFC2251] is a generic directory access protocol that is specifically targeted at management applications and browser applications that provide read/write interactive access to directories. Often such directories contain information about people, including contact/address data.

The key features of address book support in LDAP are:

1. To do.

The key disadvantages of address book support in LDAP are:

1. Lack of schemas require overly complex client configuration to map expected fields in the client to directory entries in the server.

2. General reluctance to give 'ordinary' users write access to even a small portion of the directory as often sensitive information is included in directory entries and a small mistake in configuring access control can lead to a major security breach.

1.4. SyncML

SyncML is a protocol for synchronizing data, including contacts, between different devices.

More...

1.5. WebDAV for Address Books

WebDAV [RFC4918] offers a number of advantages as a framework or basis for address book access and management. Most of these advantages boil down to a significant reduction in design costs, implementation costs, interoperability test costs and deployment costs.

The key features of address book support with WebDAV are:

1. Ability to use multiple address books with hierarchical layout.

2. Ability to control access to individual address books and address entries.

3. Principal namespace can be used to enumerate and find other users on the system.

4. Server-side searching of address data, avoiding the need for clients to download an entire address book in order to do a quick address 'expansion' operation.

5. Well-defined internationalization support through standard HTTP.

6. Use of vCards for well defined address schema to enhance client interoperability.

7. Many limited clients (e.g. mobile devices) contain an HTTP stack which makes implementing WebDAV much easier than other protocols.

The key disadvantages of address book support in WebDAV are:

1. Lack of change notification.

2. Stateless nature of protocol can result in more data being sent with each transaction to maintain per-user session across requests.

1.6. vCard

vCard [RFC2426] is a MIME directory profile aimed at encapsulating personal addressing and contact information about people. The specification of vCard was originally done by the Versit consortium, with a subsequent 3.0 version standardized by the IETF [RFC2426]. vCard is in wide spread use in email clients and mobile devices as a means of encapsulating address information for transport via email, or for import/export and synchronization operations.

2. Conventions

2.1. Notational Conventions

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].

The term "protected" is used in the Conformance field of property

definitions as defined in Section 15 of [RFC4918].

When XML element types in the namespaces "DAV:" and "urn:ietf:params:xml:ns:carddav" are referenced in this document outside of the context of an XML fragment, the string "DAV:" and "CARDDAV:" will be prefixed to the element type names, respectively.

2.2. XML Namespaces and Processing

Definitions of XML elements in this document use XML element type declarations (as found in XML Document Type Declarations), described in Section 3.2 of [W3C.REC-xml-20060816].

The namespace "urn:ietf:params:xml:ns:carddav" is reserved for the XML elements defined in this specification, its revisions, and related CardDAV specifications. XML elements defined by individual implementations MUST NOT use the "urn:ietf:params:xml:ns:carddav" namespace, and instead should use a namespace that they control.

The XML declarations used in this document do not include namespace information. Thus, implementers must not use these declarations as the only way to create valid CardDAV properties or to validate CardDAV XML element type. Some of the declarations refer to XML elements defined by WebDAV [RFC4918] which use the "DAV:" namespace. Wherever such XML elements appear, they are explicitly prefixed with "DAV:" to avoid confusion.

Also note that some CardDAV XML element names are identical to WebDAV XML element names, though their namespace differs. Care must be taken not to confuse the two sets of names.

Processing of XML by CardDAV clients and servers MUST follow the rules described in Appendix A of [RFC4918].

3. Requirements Overview

This section lists what functionality is required of a CardDAV server. To advertise support for CardDAV, a server:

- MUST support vCard [RFC2426] as a media type for the address object resource format;
- MUST support WebDAV Class 3 [RFC4918];
- MUST support WebDAV ACL [RFC3744];

- MUST support secure transport as defined in [RFC2818] using TLS v1.0 [RFC2246] or a subsequent standards-track version of TLS;

- MUST support ETags [RFC2616] with additional requirements specified in Section 6.3.2.3 of this document;

- MUST support all address book REPORTs defined in Section 8 of this document; and

- MUST advertise support on all addressbook collections and address object resources for the addressbook reports in the DAV:supported-report-set property, as defined in Versioning Extensions to WebDAV [RFC3253].

In addition, a server:

- SHOULD support the extended MKCOL method [draft-daboo-webdav-mkcol-00] to create address book collections as defined in Section 6.3.1 of this document.

4. Address Book Data Model

As a brief overview, a CardDAV address book is modeled as a WebDAV collection with a well defined structure; each of these address book collections contain a number of resources representing address objects as their direct child resources. Each resource representing an address object is called an "address object resource". Each address object resource and each address book collection can be individually locked and have individual WebDAV properties. Requirements derived from this model are provided in Section 5.1 and Section 5.2.

4.1. Address Book Server

A CardDAV server is an address-aware engine combined with a WebDAV server. The server may include address data in some parts of its URL namespace, and non-address data in other parts.

A WebDAV server can advertise itself as a CardDAV server if it supports the functionality defined in this specification at any point within the root of its repository. That might mean that address data is spread throughout the repository and mixed with non-address data in nearby collections (e.g. address data may be found in /lisa/addressbook/ as well as in /bernard/addressbook/, and non-address data in /lisa/calendars/). Or, it might mean that address data can be found only in certain sections of the repository (e.g. /addressbooks/user/). Address book features are only required in the

repository sections that are or contain address objects. So a repository confining address data to the /carddav/ collection would only need to support the CardDAV required features within that collection.

The CardDAV server is the canonical location for address data and state information. Clients may submit requests to change data or download data. Clients may store address objects offline and attempt to synchronize at a later time. However, clients MUST be prepared for address data on the server to change between the time of last synchronization and when attempting an update, as address book collections may be shared and accessible via multiple clients. Entity tags and other features help this work.

5. Address Book Resources

5.1. Address Object Resources

This specification uses vCard as the default format for address or contact information being stored on the server. However, this specification does allow other formats for address data provided that the server advertises support for those additional formats as described below. The requirements in this section pertain to vCard address data, or formats that follow the semantics of vCard data.

Address object resources contained in address book collections MUST contain a single vCard component only.

vCard components in an address book collection MUST have a UID property value that MUST be unique in the scope of the address book collection in which it is contained.

5.2. Address Book Collections

Address book collections appear to clients as a WebDAV collection resource, identified by a URL. An address book collection MUST report the DAV:collection and CARDDAV:addressbook XML elements in the value of the DAV:resourcetype property. The element type declaration for CARDDAV:addressbook is:

```
<!ELEMENT addressbook EMPTY>
```

An address book collection can be created through provisioning (e.g., automatically created when a user's account is provisioned), or it can be created with the extended MKCOL method (see Section 6.3.1). This can be used by a user to create additional address books (e.g., "soccer team members") or for users to share an address book (e.g.,

"sales team contacts"). Note however that this document doesn't define what extra address book collections are for. Users must rely on non-standard cues to find out what an address book collection is for, or use the CARDDAV:addressbook-description property defined in Section 6.2.1 to provide such a cue.

The following restrictions are applied to the resources within an address book collection:

a. Address book collections MUST only contain address object resources and collections that are not address book collections. i.e., the only "top-level" non-collection resources allowed in an address book collection are address object resources. This ensures that address book clients do not have to deal with non-address data in an address book collection, though they do have to distinguish between address object resources and collections when using standard WebDAV techniques to examine the contents of a collection.

b. Collections contained in address book collections MUST NOT contain address book collections at any depth. i.e., "nesting" of address book collections within other address book collections at any depth is not allowed. This specification does not define how collections contained in an address book collection are used or how they relate to any address object resources contained in the address book collection.

Multiple address book collections MAY be children of the same collection.

6. Address Book Feature

6.1. Address Book Support

A server supporting the features described in this document, MUST include "addressbook" as a field in the DAV response header from an OPTIONS request on any resource that supports any address book properties, reports, or methods. A value of "addressbook" in the DAV response header MUST indicate that the server supports all MUST level requirements and REQUIRED features specified in this document.

6.1.1. Example: Using OPTIONS for the Discovery of Support for CardDAV

```
>> Request <<

OPTIONS /addressbooks/users/ HTTP/1.1
Host: addressbook.example.com

>> Response <<

HTTP/1.1 200 OK
Allow: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, COPY, MOVE
Allow: MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, REPORT, ACL
DAV: 1, 2, 3, access-control, addressbook
DAV: extended-mkcol
Date: Sat, 11 Nov 2006 09:32:12 GMT
Content-Length: 0
```

In this example, the OPTIONS response indicates that the server supports CardDAV in this namespace, therefore the '/addressbooks/users/' collection may be used as a parent for address book collections as the extended MKCOL method is available, and as a possible target for REPORT requests for address book reports.

6.2. Address Book Properties

6.2.1. CARDDAV:addressbook-description Property

Name: addressbook-description

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Provides a human-readable description of the address book collection.

Value: Any text.

Protected: SHOULD NOT be protected so that users can specify a description.

COPY/MOVE behavior: This property value SHOULD be preserved in COPY and MOVE operations.

allprop behavior: SHOULD be returned by a PROPFIND DAV:allprop request.

Description: This property contains a description of the address book collection that is suitable for presentation to a user.

Definition:

```
<!ELEMENT addressbook-description (#PCDATA)>
<!-- PCDATA value: string -->
```

Example:

```
<C:addressbook-description xml:lang="fr-CA"
   xmlns:C="urn:ietf:params:xml:ns:carddav"
>Adresses de Oliver Daboo</C:addressbook-description>
```

6.2.2. CARDDAV:supported-address-data Property

Name: supported-address-data

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Specifies what media types are allowed for address object resources in an address book collection.

Protected: MUST be protected as it indicates the level of support provided by the server.

COPY/MOVE behavior: This property value MUST be preserved in COPY and MOVE operations.

allprop behavior: SHOULD be returned by a PROPFIND DAV:allprop request.

Description: The CARDDAV:supported-address-data property is used to specify the media type supported for the address object resources contained in a given address book collection (e.g., vCard version 3.0). Any attempt by the client to store address object resources with a media type not listed in this property MUST result in an error, with the CARDDAV:supported-address-data precondition (Section 6.3.2.1) being violated. In the absence of this property the server MUST only accept data with the media type "text/vcard" and vCard version 3.0, and clients can assume that.

Definition:

```
<!ELEMENT supported-address-data (address-data+)>
```

Example:

```
<C:supported-address-data
   xmlns:C="urn:ietf:params:xml:ns:carddav">
  <C:address-data content-type="text/vcard" version="3.0"/>
</C:supported-address-data>
```

6.2.3. CARDDAV:max-resource-size Property

Name: max-resource-size

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Provides a numeric value indicating the maximum size of a resource in octets that the server is willing to accept when an address object resource is stored in an address book collection.

Value: Any text representing a numeric value.

Protected: MUST be protected as it indicates limits provided by the server.

COPY/MOVE behavior: This property value MUST be preserved in COPY and MOVE operations.

allprop behavior: SHOULD be returned by a PROPFIND DAV:allprop request.

Description: The CARDDAV:max-resource-size is used to specify a numeric value that represents the maximum size in octets that the server is willing to accept when an address object resource is stored in an address book collection. Any attempt to store an address book object resource exceeding this size MUST result in an error, with the CARDDAV:max-resource-size precondition (Section 6.3.2.1) being violated. In the absence of this property the client can assume that the server will allow storing a resource of any reasonable size.

Definition:

```
<!ELEMENT max-resource-size (#PCDATA)>
<!-- PCDATA value: a numeric value (positive integer) -->
```

Example:

```
<C:max-resource-size xmlns:C="urn:ietf:params:xml:ns:carddav"
>102400</C:max-resource-size>
```

6.3. Creating Resources

Address book collections and address object resources may be created by either a CardDAV client or by the CardDAV server. This specification defines restrictions and a data model that both clients and servers MUST adhere to when manipulating such address data.

6.3.1. Extended MKCOL Method

An HTTP request using the extended MKCOL method [draft-daboo-webdav-mkcol-00] can be used to create a new address book collection resource. A server MAY restrict address book collection creation to particular collections.

To create an address book, the client sends an extended MKCOL request to the server and in the body of the request sets the DAV: resourcetype property to the resource type for an address book collection as defined in Section 5.2.

Support for creating address books on the server is only RECOMMENDED and not REQUIRED because some address book stores only support one address book per user (or principal), and those are typically pre-created for each account. However, servers and clients are strongly encouraged to support address book creation whenever possible to allow users to create multiple address book collections to help organize their data better.

Clients SHOULD use the DAV:displayname property for a human-readable name of the address book. Clients can either specify the value of the DAV:displayname property in the request body of the extended MKCOL request, or alternatively issue a PROPPATCH request to change the DAV:displayname property to the appropriate value immediately after using the extended MKCOL request. Clients SHOULD NOT set the DAV: displayname property to be the same as any other address book collection at the same URI "level". When displaying address book collections to users, clients SHOULD check the DAV:displayname property and use that value as the name of the address book. In the event that the DAV: displayname property is not set, the client MAY use the last part of the address book collection URI as the name, however that path segment may be "opaque" and not represent any meaningful human-readable text.

6.3.1.1. Example - Successful MKCOL request

This example creates an address book collection called /home/lisa/addressbook/ on the server addressbook.example.com with specific values for the properties DAV:resourcetype, DAV:displayname and CARDDAV:addressbook-description.

```
   >> Request <<

   MKCOL /home/lisa/addressbook/ HTTP/1.1
   Host: addressbook.example.com
   Content-Type: text/xml; charset="utf-8"
   Content-Length: xxx

   <?xml version="1.0" encoding="utf-8" ?>
   <D:mkcol xmlns:D="DAV:"
                xmlns:C="urn:ietf:params:xml:ns:carddav">
     <D:set>
       <D:prop>
         <D:resourcetype>
           <D:collection/>
           <C:addressbook/>
         </D:resourcetype>
         <D:displayname>Lisa's Contacts</D:displayname>
         <C:addressbook-description xml:lang="en"
   >My primary address book.</C:addressbook-description>
       </D:prop>
     </D:set>
   </D:mkcol>

   >> Response <<

   HTTP/1.1 201 Created
   Cache-Control: no-cache
   Date: Sat, 11 Nov 2006 09:32:12 GMT
   Content-Type: application/xml; charset="utf-8"
   Content-Length: xxxx

   <?xml version="1.0" encoding="utf-8" ?>
   <D:mkcol-response xmlns:D="DAV:"
                xmlns:C="urn:ietf:params:xml:ns:carddav">
     <D:propstat>
       <D:prop>
         <D:resourcetype/>
         <D:displayname/>
         <C:addressbook-description/>
       </D:prop>
       <D:status>HTTP/1.1 200 OK</D:status>
     </D:propstat>
   </D:mkcol-response>
```

6.3.2. Creating Address Object Resources

Clients populate address book collections with address object resources. The URL for each address object resource is entirely arbitrary, and does not need to bear a specific relationship (but might) to the address object resource's vCard properties or other metadata. New address object resources MUST be created with a PUT request targeted at an unmapped URI. A PUT request targeted at a mapped URI updates an existing address object resource.

When servers create new resources, it's not hard for the server to choose a unique URL. It's slightly tougher for clients, because a client might not want to examine all resources in the collection, and might not want to lock the entire collection to ensure that a new one isn't created with a name collision. However, there is an HTTP feature to mitigate this. If the client intends to create a new address resource the client SHOULD use the HTTP header "If-None-Match: *" on the PUT request. The Request-URI on the PUT request MUST include the target collection, where the resource is to be created, plus the name of the resource in the last path segment. The "If-None-Match" header ensures that the client will not inadvertently overwrite an existing resource even, if the last path segment turned out to already be used.

>> Request <<

```
PUT /lisa/addressbook/newvcard.vcf HTTP/1.1
If-None-Match: *
Host: addressbook.example.com
Content-Type: text/vcard
Content-Length: xxx

BEGIN:VCARD
VERSION:3.0
FN:Cyrus Daboo
N:Daboo;Cyrus
ADR;TYPE=POSTAL:;2822 Email HQ;Suite 2821;RFCVille;PA;15213;USA
EMAIL;TYPE=INTERNET,PREF:cyrus@example.com
NICKNAME:me
NOTE:Example VCard.
ORG:Self Employed
TEL;TYPE=WORK,VOICE:412 605 0499
TEL;TYPE=FAX:412 605 0705
URL:http://www.example.com
UID:1234-5678-9000-1
END:VCARD
```

```
>> Response <<

HTTP/1.1 201 Created
Date: Thu, 02 Sep 2004 16:53:32 GMT
Content-Length: 0
ETag: "123456789-000-111"
```

The request to change an existing address object resource is the same, but with a specific ETag in the "If-Match" header, rather than the "If-None-Match" header.

File names for vCards are commonly suffixed by ".vcf", and clients may choose to use the same convention for URLs.

6.3.2.1. Additional Preconditions for PUT, COPY and MOVE

This specification creates additional Preconditions for PUT, COPY and MOVE methods. These preconditions apply:

- When a PUT operation of an address object resource into an address book collection occurs.
- When a COPY or MOVE operation of an address object resource into an address book collection occurs.

The new preconditions are:

(CARDDAV:supported-address-data): The resource submitted in the PUT request, or targeted by a COPY or MOVE request MUST be a supported media type (i.e., vCard) for address object resources;

(CARDDAV:valid-address-data): The resource submitted in the PUT request, or targeted by a COPY or MOVE request MUST be valid data for the media type being specified (i.e., MUST contain valid vCard data);

(CARDDAV:no-uid-conflict): The resource submitted in the PUT request, or targeted by a COPY or MOVE request MUST NOT specify a vCard UID property value already in use in the targeted address book collection or overwrite an existing address object resource with one that has a different UID property value. Servers SHOULD report the URL of the resource that is already making use of the same UID property value in the DAV:href element;

```
<!ELEMENT no-uid-conflict (DAV:href)>
```

(CARDDAV:addressbook-collection-location-ok): In a COPY or MOVE request, when the Request-URI is an address book collection, the URI targeted by the Destination HTTP Request header MUST identify a location where an address book collection can be created;

(CARDDAV:max-resource-size): The resource submitted in the PUT request, or targeted by a COPY or MOVE request MUST have an octet size less than or equal to the value of the CARDDAV:max-resource-size property value (Section 6.2.3) on the address book collection where the resource will be stored;

6.3.2.2. Non-Standard vCard Properties, and Parameters

vCard provides a "standard mechanism for doing non-standard things". This extension support allows implementers to make use of non-standard properties and parameters whose names are prefixed with the text "X-".

Servers MUST support the use of non-standard properties and parameters in address object resources stored via the PUT method.

Servers may need to enforce rules for their own "private" properties or parameters, so servers MAY reject any attempt by the client to change those or use values for those outside of any restrictions the server may have. Servers SHOULD ensure that any "private" properties or parameters it uses follow the convention of including a vendor id in the "X-" name, as described in Section 3.8 of [RFC2426], e.g., "X-ABC-PRIVATE".

6.3.2.3. Address Object Resource Entity Tag

The DAV:getetag property MUST be defined and set to a strong entity tag on all address object resources.

A response to a GET request targeted at an address object resource MUST contain an ETag response header field indicating the current value of the strong entity tag of the address object resource.

Servers SHOULD return a strong entity tag (ETag header) in a PUT response when the stored address object resource is equivalent by octet equality to the address object resource submitted in the body of the PUT request. This allows clients to reliably use the returned strong entity tag for data synchronization purposes. For instance, the client can do a PROPFIND request on the stored address object resource and have the DAV:getetag property returned, and compare that value with the strong entity tag it received on the PUT response, and know that if they are equal, then the address object resource on the server has not been changed.

In the case where the data stored by a server as a result of a PUT request is not equivalent by octet equality to the submitted address object resource, the behavior of the ETag response header is not specified here, with the exception that a strong entity tag MUST NOT be returned in the response. As a result, clients may need to retrieve the modified address object resource (and ETag) as a basis for further changes, rather than use the address object resource it had sent with the PUT request.

7. Address Book Access Control

CardDAV servers MUST support and adhere to the requirements of WebDAV ACL [RFC3744]. WebDAV ACL provides a framework for an extensible set of privileges that can be applied to WebDAV collections and ordinary resources.

7.1. Additional Principal Properties

This section defines additional properties for WebDAV principal resources as defined in [RFC3744].

7.1.1. CARDDAV:addressbook-home-set Property

Name: addressbook-home-set

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Identifies the URL of any WebDAV collections that contain address book collections owned by the associated principal resource.

Protected: MAY be protected if the server has fixed locations in which address books are created.

COPY/MOVE behavior: This property value MUST be preserved in COPY and MOVE operations.

allprop behavior: SHOULD be returned by a PROPFIND DAV:allprop request.

Description: The CARDDAV:addressbook-home-set property is meant to allow users to easily find the address book collections owned by the principal. Typically, users will group all the address book collections that they own under a common collection. This property specifies the URL of collections that either are address book collections or ordinary collections that have child or descendant address book collections owned by the principal.

Definition:

```
<!ELEMENT addressbook-home-set (DAV:href*)>
```

Example:

```
<C:addressbook-home-set xmlns:D="DAV:"
   xmlns:C="urn:ietf:params:xml:ns:carddav">
  <D:href>http://addressbook.example.com/bernard/addresses/</D:href>
</C:addressbook-home-set>
```

7.1.2. CARDDAV:principal-address Property

Name: principal-address

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Identifies the URL of an address object resource that corresponds to the user represented by the principal.

Protected: MAY be protected if the server provides a fixed location for principal addresses.

COPY/MOVE behavior: This property value MUST be preserved in COPY and MOVE operations.

allprop behavior: SHOULD be returned by a PROPFIND DAV:allprop request.

Description: The CARDDAV:principal-address property is meant to allow users to easily find contact information for users represented by principals on the system. This property specifies the URL of the address object resource containing the corresponding contact information.

Definition:

```
<!ELEMENT principal-address (DAV:href)>
```

Example:

```
<C:principal-address xmlns:D="DAV:"
   xmlns:C="urn:ietf:params:xml:ns:carddav">
  <D:href>http://addressbook.example.com/system/cyrus.vcf</D:href>
</C:principal-address>
```

8. Address Book Reports

This section defines the reports that CardDAV servers MUST support on address book collections and address object resources.

CardDAV servers MUST advertise support for these REPORTs on all address book collections and address object resources with the DAV: supported-report-set property defined in Section 3.1.5 of [RFC3253]. CardDAV servers MAY also advertise support for these REPORTs on ordinary collections.

Some of these REPORTs allow address data (from possibly multiple resources) to be returned.

8.1. REPORT Method

The REPORT method (defined in Section 3.6 of [RFC3253]) provides an extensible mechanism for obtaining information about a resource. Unlike the PROPFIND method, which returns the value of one or more named properties, the REPORT method can involve more complex processing. REPORT is valuable in cases where the server has access to all of the information needed to perform the complex request (such as a query), and where it would require multiple requests for the client to retrieve the information needed to perform the same request.

A server that supports this specification MUST support the DAV: expand-property report (defined in Section 3.8 of [RFC3253]).

8.2. Ordinary collections

Servers MAY support the REPORTs defined in this document on ordinary collections (collections that are not address book collections) in addition to address book collections or address object resources. In computing responses to the REPORTs on ordinary collections, servers MUST only consider address object resources contained in address book collections that are targeted by the REPORT based on the value of the Depth request header.

8.3. Searching Text: Collations

Some of the reports defined in this section do text matches of character strings provided by the client and compared to stored address data. Since vCard data is by default encoded in the UTF-8 charset and may include characters outside of the US-ASCII charset range in some property and parameter values, there is a need to ensure that text matching follows well-defined rules.

To deal with this, this specification makes use of the IANA Collation Registry defined in [RFC4790] to specify collations that may be used to carry out the text comparison operations with a well-defined rule.

Collations supported by the server MUST support "equality" and "substring" match operations as per [RFC4790] Section 4.2, including the "prefix" and "suffix" options for "substring" matching. CardDAV uses these match options for "equals", "contains", "starts-with" and "ends-with" match operations.

CardDAV servers are REQUIRED to support the "i;ascii-casemap" [RFC4790] and "i;unicode-casemap" [RFC5051] collations, and MAY support other collations.

Servers MUST advertise the set of collations that they support via the CARDDAV:supported-collation-set property defined on any resource that supports reports that use collations.

In the absence of a collation explicitly specified by the client, or if the client specifies the "default" collation identifier (as defined in [RFC4790] Section 3.1), the server MUST default to using "i;unicode-casemap" as the collation.

Wildcards (as defined in [RFC4790] Section 3.2) MUST NOT be used in the collation identifier.

If the client chooses a collation not supported by the server, the server MUST respond with a CARDDAV:supported-collation precondition error response.

8.3.1. CARDDAV:supported-collation-set Property

Name: supported-collation-set

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Identifies the set of collations supported by the server for text matching operations.

Protected: MUST be protected as it indicates support provided by the server.

COPY/MOVE behavior: This property value MUST be preserved in COPY and MOVE operations.

allprop behavior: SHOULD be returned by a PROPFIND DAV:allprop request.

Description: The CARDDAV:supported-collation-set property contains zero or more CARDDAV:supported-collation elements which specify the collection identifiers of the collations supported by the server.

Definition:

```
<!ELEMENT supported-collation-set (supported-collation*)>

<!ELEMENT supported-collation (#PCDATA)>
```

Example:

```
<C:supported-collation-set
  xmlns:C="urn:ietf:params:xml:ns:carddav">
  <C:supported-collation>i;ascii-casemap</C:supported-collation>
  <C:supported-collation>i;octet</C:supported-collation>
  <C:supported-collation>i;unicode-casemap</C:supported-collation>
</C:supported-collation-set>
```

8.4. Partial Retrieval

Some address book REPORTs defined in this document allow partial retrieval of address object resources. A CardDAV client can specify what information to return in the body of an address book REPORT request.

A CardDAV client can request particular WebDAV property values, all WebDAV property values, or a list of the names of the resource's WebDAV properties. A CardDAV client can also request address data to be returned and whether all vCard properties should be returned or only particular ones. See CARDDAV:address-data in Section 10.4.

8.5. Non-standard properties and parameters

Servers MUST support the use of non-standard property or parameter names in the CARDDAV:address-data XML element in address book REPORT requests to allow clients to request that non-standard properties and parameters be returned in the address data provided in the response.

Servers MAY support the use of non-standard property or parameter names in the CARDDAV:prop-filter and CARDDAV:param-filter XML elements specified in the CARDDAV:filter XML element of address book REPORT requests.

Servers MUST fail with the CARDDAV:supported-filter precondition if an address book REPORT request uses a CARDDAV:prop-filter or CARDDAV:param-filter XML element that makes reference to a non-standard property or parameter name which the server does not support queries on.

8.6. CARDDAV:addressbook-query Report

The CARDDAV:addressbook-query REPORT performs a search for all address object resources that match a specified filter. The response of this REPORT will contain all the WebDAV properties and address object resource data specified in the request. In the case of the CARDDAV:address-data XML element, one can explicitly specify the vCard properties that should be returned in the address object resource data that matches the filter.

The format of this report is modeled on the PROPFIND method. The request and response bodies of the CARDAV:addressbook-query report use XML elements that are also used by PROPFIND. In particular the request can include XML elements to request WebDAV properties to be returned. When that occurs the response should follow the same behavior as PROPFIND with respect to the DAV:multistatus response elements used to return specific property results. For instance, a request to retrieve the value of a property which does not exist is an error and MUST be noted with a response XML element which contains a 404 (Not Found) status value.

Support for the CARDDAV:addressbook-query REPORT is REQUIRED.

Marshalling:

The request body MUST be a CARDDAV:addressbook-query XML element as defined in Section 10.3.

The request MAY include a Depth header. If no Depth header is included, Depth:0 is assumed.

The response body for a successful request MUST be a DAV:multistatus XML element (i.e., the response uses the same format as the response for PROPFIND). In the case where there are no response elements, the returned DAV:multistatus XML element is empty.

The response body for a successful CARDDAV:addressbook-query REPORT request MUST contain a DAV:response element for each address object that matched the search filter. address data is returned in the CARDDAV:address-data XML element inside the DAV:propstat XML element.

Preconditions:

(CARDDAV:supported-address-data): The attributes "content-type" and "version" of the CARDDAV:address-data XML element (see Section 10.4) specify a media type supported by the server for address object resources.

(CARDDAV:supported-filter): The CARDDAV:prop-filter (see Section 10.5.1) and CARDDAV:param-filter (see Section 10.5.2) XML elements used in the CARDDAV:filter XML element (see Section 10.5) in the REPORT request only make reference to properties and parameters for which queries are supported by the server. i.e., if the CARDDAV:filter element attempts to reference an unsupported property or parameter, this precondition is violated. Servers SHOULD report the CARDDAV:prop-filter or CARDDAV:param-filter for which it does not provide support.

```
<!ELEMENT supported-filter (prop-filter*,
                            param-filter*)>
```

(CARDDAV:supported-collation): Any XML attribute specifying a collation MUST specify a collation supported by the server as described in Section 8.3.

Postconditions:

(DAV:number-of-matches-within-limits): The number of matching address object resources must fall within server-specific, predefined limits. For example, this condition might be triggered if a search specification would cause the return of an extremely large number of responses.

8.6.1. Example: Partial retrieval of vCards matching a NICKNAME

In this example, the client requests the server to search for address object resources that contain a NICKNAME property whose value equals some specific text, and to return specific vCard properties for those vCards found. In addition the DAV:getetag property is also requested and returned as part of the response.

>> Request <<

```
REPORT /home/bernard/addressbook/ HTTP/1.1
Host: addressbook.example.com
Depth: 1
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<C:addressbook-query xmlns:D="DAV:"
                 xmlns:C="urn:ietf:params:xml:ns:carddav">
  <D:prop>
    <D:getetag/>
    <C:address-data>
      <C:prop name="VERSION"/>
      <C:prop name="UID"/>
      <C:prop name="NICKNAME"/>
      <C:prop name="EMAIL"/>
      <C:prop name="FN"/>
    </C:address-data>
  </D:prop>
  <C:filter>
    <C:prop-filter name="NICKNAME">
      <C:text-match collation="i;unicode-casemap"
                    match-type="equals"
      >me</C:text-match>
    </C:prop-filter>
  </C:filter>
</C:addressbook-query>
```

```
   >> Response <<


   HTTP/1.1 207 Multi-Status
   Date: Sat, 11 Nov 2006 09:32:12 GMT
   Content-Type: text/xml; charset="utf-8"
   Content-Length: xxxx

   <?xml version="1.0" encoding="utf-8" ?>
   <D:multistatus xmlns:D="DAV:"
                  xmlns:C="urn:ietf:params:xml:ns:carddav">
     <D:response>
       <D:href>/home/bernard/addressbook/v102.vcf</D:href>
       <D:propstat>
         <D:prop>
           <D:getetag>"23ba4d-ff11fb"</D:getetag>
           <C:address-data>BEGIN:VCARD
   VERSION:3.0
   NICKNAME:me
   UID:34222-232@example.com
   FN:Cyrus Daboo
   EMAIL:daboo@example.com
   END:VCARD
   </C:address-data>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
       </D:propstat>
     </D:response>
   </D:multistatus>
```

8.6.2. Example: Partial retrieval of vCards matching a full name

In this example, the client requests the server to search for address object resources that contain a FN property whose value contains some specific text, and to return specific vCard properties for those vCards found. In addition the DAV:getetag property is also requested and returned as part of the response.

>> Request <<

```
REPORT /home/bernard/addressbook/ HTTP/1.1
Host: addressbook.example.com
Depth: 1
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<C:addressbook-query xmlns:D="DAV:"
                  xmlns:C="urn:ietf:params:xml:ns:carddav">
  <D:prop>
    <D:getetag/>
    <C:address-data>
      <C:prop name="VERSION"/>
      <C:prop name="UID"/>
      <C:prop name="NICKNAME"/>
      <C:prop name="EMAIL"/>
      <C:prop name="FN"/>
    </C:address-data>
  </D:prop>
  <C:filter>
    <C:prop-filter name="FN">
      <C:text-match collation="i;unicode-casemap"
                    match-type="contains"
      >Daboo</C:text-match>
    </C:prop-filter>
  </C:filter>
</C:addressbook-query>
```

```
   >> Response <<


   HTTP/1.1 207 Multi-Status
   Date: Sat, 11 Nov 2006 09:32:12 GMT
   Content-Type: text/xml; charset="utf-8"
   Content-Length: xxxx

   <?xml version="1.0" encoding="utf-8" ?>
   <D:multistatus xmlns:D="DAV:"
                  xmlns:C="urn:ietf:params:xml:ns:carddav">
     <D:response>
       <D:href>/home/bernard/addressbook/v102.vcf</D:href>
       <D:propstat>
         <D:prop>
           <D:getetag>"23ba4d-ff11fb"</D:getetag>
           <C:address-data>BEGIN:VCARD
   VERSION:3.0
   NICKNAME:me
   UID:34222-232@example.com
   FN:Cyrus Daboo
   EMAIL:daboo@example.com
   END:VCARD
   </C:address-data>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
       </D:propstat>
     </D:response>
     <D:response>
       <D:href>/home/bernard/addressbook/v104.vcf</D:href>
       <D:propstat>
         <D:prop>
           <D:getetag>"23ba4d-ff11fc"</D:getetag>
           <C:address-data>BEGIN:VCARD
   VERSION:3.0
   NICKNAME:oliver
   UID:34222-23222@example.com
   FN:Oliver Daboo
   EMAIL:oliver@example.com
   END:VCARD
   </C:address-data>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
       </D:propstat>
     </D:response>
   </D:multistatus>
```

8.7. CARDDAV:addressbook-multiget Report

The CARDDAV:addressbook-multiget REPORT is used to retrieve specific address object resources from within a collection, if the Request-URI is a collection, or to retrieve a specific address object resource, if the Request-URI is a address object resource. This report is similar to the CARDDAV:addressbook-query REPORT (see Section 8.6), except that it takes a list of DAV:href elements instead of a CARDDAV:filter element to determine which address object resources to return.

Support for the addressbook-multiget REPORT is REQUIRED.

Marshalling:

The request body MUST be a CARDDAV:addressbook-multiget XML element (see Section 10.6, which MUST contain at least one DAV: href XML element, and one optional CARDDAV:address-data element as defined in Section 10.4. If the Request-URI is a collection resource, then the DAV:href elements MUST refer to resources within that collection, and they MAY refer to resources at any depth within the collection. As a result the "Depth" header MUST be ignored by the server and SHOULD NOT be sent by the client. If the Request-URI refers to a non-collection resource, then there MUST be a single DAV:href element that is equivalent to the Request-URI.

The response body for a successful request MUST be a DAV: multistatus XML element.

The response body for a successful CARDDAV:addressbook-multiget REPORT request MUST contain a DAV:response element for each address object resource referenced by the provided set of DAV:href elements. Address data is returned in the CARDDAV:address-data element inside the DAV:prop element.

In the case of an error accessing any of the provided DAV:href resources, the server MUST return the appropriate error status code in the DAV:status element of the corresponding DAV:response element.

Preconditions:

(CARDAV:supported-address-data): The attributes "content-type" and "version" of the CARDDAV:address-data XML elements (see Section 10.4) specify a media type supported by the server for address object resources.

Postconditions:

None.

8.7.1. Example: CARDDAV:addressbook-multiget Report

In this example, the client requests the server to return specific properties of the address components referenced by specific URIs. In addition the DAV:getetag property is also requested and returned as part of the response. Note that in this example, the resource at http://addressbook.example.com/home/bernard/addressbook/vcf1.vcf does not exist, resulting in an error status response.

>> Request <<

```
REPORT /home/bernard/addressbook/ HTTP/1.1
Host: addressbook.example.com
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<C:addressbook-multiget xmlns:D="DAV:"
                     xmlns:C="urn:ietf:params:xml:ns:carddav">
  <D:prop>
    <D:getetag/>
    <C:address-data>
      <C:prop name="VERSION"/>
      <C:prop name="UID"/>
      <C:prop name="NICKNAME"/>
      <C:prop name="EMAIL"/>
      <C:prop name="FN"/>
    </C:address-data>
  </D:prop>
  <D:href>/home/bernard/addressbook/vcf102.vcf</D:href>
  <D:href>/home/bernard/addressbook/vcf1.vcf</D:href>
</C:addressbook-multiget>
```

```
   >> Response <<


   HTTP/1.1 207 Multi-Status
   Date: Sat, 11 Nov 2006 09:32:12 GMT
   Content-Type: text/xml; charset="utf-8"
   Content-Length: xxxx

   <?xml version="1.0" encoding="utf-8" ?>
   <D:multistatus xmlns:D="DAV:"
                  xmlns:C="urn:ietf:params:xml:ns:carddav">
     <D:response>
       <D:href>/home/bernard/addressbook/vcf102.vcf</D:href>
       <D:propstat>
         <D:prop>
           <D:getetag>"23ba4d-ff11fb"</D:getetag>
           <C:address-data>BEGIN:VCARD
   VERSION:3.0
   NICKNAME:me
   UID:34222-232@example.com
   FN:Cyrus Daboo
   EMAIL:daboo@example.com
   END:VCARD
   </C:address-data>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
       </D:propstat>
     </D:response>
     <D:response>
       <D:href>/home/bernard/addressbook/vcf1.vcf</D:href>
       <D:status>HTTP/1.1 404 Resource not found</D:status>
     </D:response>
   </D:multistatus>
```

9. Guidelines

9.1. Restrict the Properties Returned

Clients may not need all the properties in a vCard object when presenting information to the user, or looking up specific items for their email address, for example. Since some property data can be large (e.g., PHOTO or SOUND with inline content) clients can choose to ignore those by only requesting the specific items it knows it will use, through use of the CARDDAV:address-data XML element in the relevant reports.

However, if a client needs to make a change to a vCard, it can only change the entire vCard data via a PUT request. There is no way to incrementally make a change to a set of properties within a vCard object resource. As a result the client will have to cache the entire set of properties on a resource that is being changed.

9.2. Use of Locking

WebDAV locks can be used to prevent two clients modifying the same resource from either overwriting each others' changes (though that problem can also be solved by using ETags) and also to prevent the user from making changes that will conflict with another set of changes. In a multi-user address book system, the address book client could lock an address object resource while the user is editing the vCard data, and unlock the address object resource when the user finishes or cancels. Locks can also be used to prevent changes while data is being reorganized. For example, an address book client might lock two address book collections prior to moving a bunch of address object resources from one to another.

Clients may request a lock timeout period that is appropriate to the use case. When the user explicitly decides to reserve a resource and prevent other changes, a long timeout might be appropriate, but in cases when the client automatically decides to lock the resource the timeout should be short (and the client can always refresh the lock should it need to). A short lock timeout means that if the client is unable to remove the lock, the other address book users aren't prevented from making changes.

9.3. Finding address books

Much of the time an address book client (or agent) will discover a new address book's location by being provided directly with the URL. E.g. a user will type his or her own address book location into client configuration information, or cut and paste a URL from email into the address book application. The client need only confirm that the URL points to a resource which is an address book. The client may also be able to browse WebDAV collections to find address book collections.

The choice of HTTP URLs means that address object resources are backward compatible with existing software, but does have the disadvantage that existing software does not usually know to look at the OPTIONS response to that URL to determine what can be done with it. This is somewhat of a barrier for WebDAV usage as well as with CardDAV usage. This specification does not offer a way through this other than making the information available in the OPTIONS response should this be requested.

For address book sharing use cases, one might wish to find the address book belonging to another user. If the other user has an address book in the same repository, that address book can be found by using the principal namespace required by WebDAV ACL support.

Because CardDAV requires servers to support WebDAV ACL [RFC3744] including principal namespaces, and with the addition of the CARDDAV: addressbook-home-set property, there are a couple options for CardDAV clients to find one's own address book or another user's address book.

In this case, a DAV:principal-match REPORT is used to find a named property (the CARDDAV:addressbook-home-set) on the Principal-URL of the current user. Using this, a WebDAV client can learn "who am I" and "where are my address books". The REPORT request body looks like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<D:principal-match xmlns:D="DAV:">
  <D:self/>
  <D:prop>
    <C:addressbook-home-set
       xmlns:C="urn:ietf:params:xml:ns:carddav"/>
  </D:prop>
</D:principal-match>
```

To find other users' address books, the DAV:principal-property-search REPORT can be used to filter on some properties and return others. To search for an address book owned by a user named "Laurie", the REPORT request body would look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<D:principal-property-search xmlns:D="DAV:">
  <D:property-search>
    <D:prop>
      <D:displayname/>
    </D:prop>
    <D:match>Laurie</D:match>
  </D:property-search>
  <D:prop>
    <C:addressbook-home-set
       xmlns:C="urn:ietf:params:xml:ns:carddav"/>
    <D:displayname/>
  </D:prop>
</D:principal-property-search>
```

The server performs a case-sensitive or caseless search for a matching string subset of "Laurie" within the DAV:displayname

property. Thus, the server might return "Laurie Dusseault", "Laurier Desruisseaux" or "Wilfrid Laurier" all as matching DAV:displayname values, and the address books for each of these.

10. XML Element Definitions

10.1. CARDDAV:addressbook XML Element

Name: addressbook

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Specifies the resource type of an address book collection.

Description: See Section 5.2.

Definition:

```
<!ELEMENT addressbook EMPTY>
```

10.2. CARDDAV:supported-collation XML Element

Name: supported-collation

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Identifies a single collation via its collation identifier as defined by [RFC4790].

Description: The CARDDAV:supported-collation contains the text of a collation identifier as described in Section 8.3.1.

Definition:

```
<!ELEMENT supported-collation (#PCDATA)>
<!-- PCDATA value: collation identifier -->
```

10.3. CARDDAV:addressbook-query XML Element

Name: addressbook-query

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Defines a report for querying address book data

Description: See Section 8.6.

Definition:

```
<!ELEMENT addressbook-query ((DAV:allprop |
                              DAV:propname |
                              DAV:prop)?, filter)>
```

10.4. CARDDAV:address-data XML Element

Name: address-data

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Specifies one of the following:

1. A supported media type for address object resources when nested in the CARDDAV:supported-address-data property;

2. The parts of an address object resource should be returned by a given address book REPORT;

3. The content of an address object resource in a response to an address book REPORT.

Description: When nested in the CARDDAV:supported-address-data property, the CARDDAV:address-data XML element specifies a media type supported by the CardDAV server for address object resources.

When used in an address book REPORT request, the CARDDAV:address-data XML element specifies which parts of address object resources need to be returned in the response. If the CARDDAV:address-data XML element doesn't contain any CARDDAV:prop elements, address object resources will be returned in their entirety.

Finally, when used in an address book REPORT response, the CARDDAV:address-data XML element specifies the content of a address object resource. Given that XML parsers normalize the two-character sequence CRLF (US-ASCII decimal 13 and US-ASCII decimal 10) to a single LF character (US-ASCII decimal 10), the CR character (US-ASCII decimal 13) MAY be omitted in address object resources specified in the CARDDAV:address-data XML element. Furthermore, address object resources specified in the CARDDAV:address-data XML element MAY be invalid per their media type specification if the CARDAV:address-data XML element part of the address book REPORT request did not specify required properties (e.g., UID, etc.) or specified a CARDDAV:prop XML element with the "novalue" attribute set to "yes".

Note: The CARDDAV:address-data XML element is specified in requests and responses inside the DAV:prop XML element as if it were a WebDAV property. However, the CARDDAV:address-data XML element is not a WebDAV property and as such it is not returned in PROPFIND responses nor used in PROPPATCH requests.

Note: The address data embedded within the CARDDAV:address-data XML element MUST follow the standard XML character data encoding rules, including use of <, >, & etc entity encoding or the use of a <![CDATA[...]]> construct. In the later case the vCard data cannot contain the character sequence "]]>" which is the end delimiter for the CDATA section.

Definition:

```
<!ELEMENT address-data EMPTY>

when nested in the CARDDAV:supported-address-data property
to specify a supported media type for address object
resources;

<!ELEMENT address-data (allprop | prop*)>

when nested in the DAV:prop XML element in an addressbook
REPORT request to specify which parts of address object
resources should be returned in the response;

<!ELEMENT address-data (#PCDATA)>
<!-- PCDATA value: address data -->

when nested in the DAV:prop XML element in an addressbook
REPORT response to specify the content of a returned
address object resource.

<!ATTLIST address-data content-type CDATA "text/vcard"
                       version CDATA "3.0">
<!-- content-type value: a MIME media type -->
<!-- version value: a version string -->

attributes can be used on all three variants of the
CALDAV:address-data XML element.
```

10.4.1. CARDDAV:allprop XML Element

Name: allprop

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Specifies that all properties shall be returned.

Description: This element can be used when the client wants all properties of components returned by a report.

Definition:

```
<!ELEMENT allprop EMPTY>
```

NOTE: The CARDDAV:allprop element defined here has the same name as the DAV:allprop element defined in WebDAV. However, the CARDDAV: allprop element defined here uses the "urn:ietf:params:xml:ns:carddav" namespace, as opposed to the "DAV:" namespace used for the DAV:allprop element defined in WebDAV.

10.4.2. CARDDAV:prop XML Element

Name: prop

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Defines which properties to return in the response.

Description: The "name" attribute specifies the name of the addressbook property to return (e.g., "NICKNAME"). The "novalue" attribute can be used by clients to request that the actual value of the property not be returned (if the "novalue" attribute is set to "yes"). In that case the server will return just the vCard property name and any vCard parameters and a trailing ":" without the subsequent value data.

Definition:

```
<!ELEMENT prop EMPTY>

<!ATTLIST prop name CDATA #REQUIRED
          novalue (yes | no) "no">
<!-- name value: a vCard property name -->
<!-- novalue value: "yes" or "no" -->
```

NOTE: The CARDDAV:prop element defined here has the same name as the DAV:prop element defined in WebDAV. However, the CARDDAV:prop element defined here uses the "urn:ietf:params:xml:ns:carddav" namespace, as opposed to the "DAV:" namespace used for the DAV:prop

element defined in WebDAV.

10.5. CARDDAV:filter XML Element

Name: filter

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Determines which matching objects are returned.

Description: The "filter" element specifies the search filter used to match address objects that should be returned by a report.

Definition:

```
<!ELEMENT filter (prop-filter*)>
```

10.5.1. CARDDAV:prop-filter XML Element

Name: prop-filter

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Limits the search to specific properties.

Description: The CARDDAV:prop-filter XML element specifies a search criteria on a specific vCard property (e.g., NICKNAME). An address object is said to match a CARDDAV:prop-filter if:

* A property of the type specified by the "name" attribute exists, and the CARDDAV:prop-filter is empty, or it matches the CARDDAV:text-match conditions if specified, and that any CARDDAV:param-filter child elements also match.

or:

* A property of the type specified by the "name" attribute does not exist, and the CARDAV:is-not-defined element is specified.

Definition:

```
<!ELEMENT prop-filter (is-not-defined |
                       (text-match?, param-filter*))>

<!ATTLIST prop-filter name CDATA #REQUIRED>
<!-- name value: a vCard property name (e.g., "NICKNAME") -->
```

10.5.2. CARDDAV:param-filter XML Element

Name: param-filter

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Limits the search to specific parameter values.

Description: The CARDDAV:param-filter XML element specifies a search criteria on a specific vCard property parameter (e.g., TYPE) in the scope of a given CARDDAV:prop-filter. A vCard property is said to match a CARDDAV:param-filter if:

* A parameter of the type specified by the "name" attribute exists, and the CARDDAV:param-filter is empty, or it matches the CARDDAV:text-match conditions if specified.

or:

* A parameter of the type specified by the "name" attribute does not exist, and the CARDDAV:is-not-defined element is specified.

Definition:

```
<!ELEMENT param-filter (is-not-defined | text-match)?>

<!ATTLIST param-filter name CDATA #REQUIRED>
<!-- name value: a property parameter name (e.g., "TYPE") -->
```

10.5.3. CARDDAV:is-not-defined XML Element

Name: is-not-defined

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Specifies that a match should occur if the enclosing property or parameter does not exist.

Description: The CARDDAV:is-not-defined XML element specifies that a match occurs if the enclosing property or parameter value specified in an address book REPORT request does not exist in the address data being tested.

Definition:

```
<!ELEMENT is-not-defined EMPTY>
```

10.5.4. CARDDAV:text-match XML Element

Name: text-match

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: Specifies a substring match on a property or parameter value.

Description: The CARDDAV:text-match XML element specifies text used for a substring match against the property or parameter value specified in an address book REPORT request.

The "collation" attribute is used to select the collation that the server MUST use for character string matching. In the absence of this attribute the server MUST use the "i;unicode-casemap" collation.

The "negate-condition" attribute is used to indicate that this test returns a match if the text matches, when the attribute value is set to "no", or return a match if the text does not match, if the attribute value is set to "yes". For example, this can be used to match components with a CATEGORIES property not set to PERSON.

The "match-type" attribute is used to indicate the type of match operation to use. Possible choices are:

"equals" - an exact match to the target string

"contains" - a substring match, matching anywhere within the target string

"starts-with" - a substring match, matching only at the start of the target string

"ends-with" - a substring match, matching only at the end of the target string

Definition:

```
<!ELEMENT text-match (#PCDATA)>
<!-- PCDATA value: string -->

<!ATTLIST text-match
   collation        CDATA "i;unicode-casemap"
   negate-condition (yes | no) "no"
   match-type   (equals|contains|starts-with|ends-with) "contains">
```

10.6. CARDDAV:addressbook-multiget XML Element

Name: addressbook-multiget

Namespace: urn:ietf:params:xml:ns:carddav

Purpose: CardDAV report used to retrieve specific address objects via their URIs.

Description: See Section 8.7.

Definition:

```
<!ELEMENT addressbook-multiget ((DAV:allprop |
                                 DAV:propname |
                                 DAV:prop)?,
                                DAV:href+)>
```

11. Internationalization Considerations

CardDAV allows internationalized strings to be stored and retrieved for the description of address book collections (see Section 6.2.1).

The CARDDAV:addressbook-query report (Section 8.6) includes a text searching option controlled by the CARDDAV:text-match element and details of character handling are covered in the description of that element (see Section 10.5.4).

12. Security Considerations

HTTP protocol transactions are sent in the clear over the network unless protection from snooping is negotiated. This can be accomplished by use of TLS as defined in [RFC2818]. In particular, if HTTP Basic authentication is available, the server MUST allow TLS to be used at the same time, and SHOULD prevent use of Basic authentication when TLS is not in use.

With the ACL extension present, WebDAV allows control over who can access (read or write) any resource on the WebDAV server. In addition, WebDAV ACL provides for an "inheritance" mechanism, whereby resources may inherit access privileges from other resources. Often the "other" resource is a parent collection of the resource itself. Clients MUST take care to ensure users are aware of which address books may be "private" (i.e. only accessible to them) and which are "shared" (i.e. accessible to others).

Since webservers are often the target of automated indexing applications that gather data from the server, analyze it and extract 'interesting' parts, great care must be taken when allowing unauthenticated access to any address book or address object data. Clients MAY choose to warn users when they create address data in a public address book, copy or move address data into public address books, or change access privileges in such a way as to expose address data to unauthenticated users.

This specification currently relies on standard HTTP authentication mechanisms for identifying users. These comprise Basic and Digest authentication as well as SSL using client-side certificates.

13. IANA Consideration

In addition to the namespaces defined by RFC4918 [RFC4918] for XML elements, this document uses a URN to describe a new XML namespace conforming to a registry mechanism described in RFC3688 [RFC3688]. All other IANA considerations mentioned in RFC4918 [RFC4918] also apply to this document.

13.1. Namespace Registration

Registration request for the carddav namespace:

URI: urn:ietf:params:xml:ns:carddav

Registrant Contact: See the "Author's Address" section of this document.

XML: None. Namespace URIs do not represent an XML specification.

14. Acknowledgments

Thanks go to Lisa Dusseault and Bernard Desruisseaux for their work on CalDAV, on which CardDAV is heavily based. The following individuals contributed their ideas and support for writing this specification: Stefan Eissing, Arnaud Quillaud, Julian Reschke, Elias Sinderson, Greg Stein, Wilfredo Sanchez Vega.

15. References

15.1. Normative References

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

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2426] Dawson, F. and T. Howes, "vCard MIME Directory Profile", RFC 2426, September 1998.

[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.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[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.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

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

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

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

[RFC5051] Crispin, M., "i;unicode-casemap - Simple Unicode Collation Algorithm", RFC 5051, October 2007.

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

[draft-daboo-webdav-mkcol-00] Daboo, C., "Extended MKCOL for WebDAV", November 2007.

15.2. Informative References

[IMSP] Myers, J., "IMSP - Internet Message Support Protocol", June 1995.

[RFC2244] Newman, C. and J. Myers, "ACAP -- Application Configuration Access Protocol", RFC 2244, November 1997.

[RFC2251] Wahl, M., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997.

Appendix A. Change History (to be removed prior to publication as an RFC)

Changes from -03

1. Renamed addressbook-data to address-data for consistency.

2. Fixed address-data element definition.

Changes from -02

1. Replaced MKADDRESSBOOK with extended MKCOL.

2. Now require i;uncide-casemap as a supproted collation and make it the default.

3. No longer require i;octet as a supproted collation.

4. Allow different types of match operations via the "match-type" attribute on the "text-match" element.

5. Updated to 4918 reference and removed some text/sections duplicating 4918.

6. WebDAV Level 3 now required.

7. TLS requirement text tweaked to match latest text approved by IESG.

8. Added principal-address proeprty to principal resources to allow a vcard to be associated with a principal.

9. XML definition clean-up.

Changes from -01

1. Added commentary on SyncML.

2. Changed 'adbk' to 'addressbook'.

3. Support for MKADDRESSBOOK is now a SHOULD.

4. Updated to RFC4790 reference.

5. Removed synchronization report.

6. Removed BNF conventions section as we have no BNF.

7. Reworded and reformatted several items to match the final CalDAV spec.

8. Added section on use of nonstandard proeprties and parameters (as per CalDAV).

9. Added section of behavior of ETags (as per CalDAV).

10. Generalized the text so that vCard need not be the only format supported by the server (i.e., allow xml version of vCard etc).

11. Renamed supported-addressbook-data to supported-address-data.

12. Renamed valid-addressbook-data tp valid-address-data.

13. Now requires "i;unicasemao" collation.

Changes from -00

1. Fixed various incorrect references and typos.

2. Major changes to sync with latest CalDAV spec behaviors.

Author's Address

Cyrus Daboo
Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
USA

Email: cyrus@daboo.name
URI: http://www.apple.com/

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).