

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: May 13, 2010

C. Daboo  
Apple  
November 9, 2009

vCard Extensions to WebDAV (CardDAV)  
draft-ietf-vcarddav-carddav-10

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79. This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on May 13, 2010.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

Provisions Relating to IETF Documents in effect on the date of publication of this document (<http://trustee.ietf.org/license-info>). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

## 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.

## Table of Contents

1. Introduction and Overview . . . . .	4
2. Conventions . . . . .	5
3. Requirements Overview . . . . .	6
4. Address Book Data Model . . . . .	7
4.1. Address Book Server . . . . .	7
5. Address Book Resources . . . . .	7
5.1. Address Object Resources . . . . .	8
5.1.1. Data Type Conversion . . . . .	8
5.1.1.1. Additional Precondition for GET . . . . .	9
5.2. Address Book Collections . . . . .	9
6. Address Book Feature . . . . .	10
6.1. Address Book Support . . . . .	10
6.1.1. Example: Using OPTIONS for the Discovery of Support for CardDAV . . . . .	10
6.2. Address Book Properties . . . . .	10
6.2.1. CARDDAV:addressbook-description Property . . . . .	11
6.2.2. CARDDAV:supported-address-data Property . . . . .	11
6.2.3. CARDDAV:max-resource-size Property . . . . .	12
6.3. Creating Resources . . . . .	13
6.3.1. Extended MKCOL Method . . . . .	13
6.3.1.1. Example - Successful MKCOL request . . . . .	14
6.3.2. Creating Address Object Resources . . . . .	16
6.3.2.1. Additional Preconditions for PUT, COPY and MOVE . . . . .	17
6.3.2.2. Non-Standard vCard Properties, and Parameters . . . . .	18
6.3.2.3. Address Object Resource Entity Tag . . . . .	18
7. Address Book Access Control . . . . .	19
7.1. Additional Principal Properties . . . . .	19
7.1.1. CARDDAV:addressbook-home-set Property . . . . .	19
7.1.2. CARDDAV:principal-address Property . . . . .	20
8. Address Book Reports . . . . .	21
8.1. REPORT Method . . . . .	21
8.2. Ordinary Collections . . . . .	21
8.3. Searching Text: Collations . . . . .	21
8.3.1. CARDDAV:supported-collation-set Property . . . . .	22

8.4.	Partial Retrieval . . . . .	23
8.5.	Non-standard Properties and Parameters . . . . .	23
8.6.	CARDDAV:addressbook-query Report . . . . .	24
8.6.1.	Limiting Results . . . . .	25
8.6.2.	Truncation of Results . . . . .	26
8.6.3.	Example: Partial Retrieval of vCards Matching NICKNAME . . . . .	26
8.6.4.	Example: Partial Retrieval of vCards Matching a Full Name or Email Address . . . . .	28
8.6.5.	Example: Truncated Results . . . . .	31
8.7.	CARDDAV:addressbook-multiget Report . . . . .	32
8.7.1.	Example: CARDDAV:addressbook-multiget Report . . . . .	34
8.7.2.	Example: CARDDAV:addressbook-multiget Report . . . . .	35
9.	Client Guidelines . . . . .	36
9.1.	Restrict the Properties Returned . . . . .	36
9.2.	Avoiding Lost Updates . . . . .	37
9.3.	Client Configuration . . . . .	37
9.4.	Finding Other Users' Address Books . . . . .	37
10.	XML Element Definitions . . . . .	38
10.1.	CARDDAV:addressbook XML Element . . . . .	38
10.2.	CARDDAV:supported-collation XML Element . . . . .	38
10.3.	CARDDAV:addressbook-query XML Element . . . . .	39
10.4.	CARDDAV:address-data XML Element . . . . .	39
10.4.1.	CARDDAV:allprop XML Element . . . . .	41
10.4.2.	CARDDAV:prop XML Element . . . . .	41
10.5.	CARDDAV:filter XML Element . . . . .	42
10.5.1.	CARDDAV:prop-filter XML Element . . . . .	43
10.5.2.	CARDDAV:param-filter XML Element . . . . .	44
10.5.3.	CARDDAV:is-not-defined XML Element . . . . .	45
10.5.4.	CARDDAV:text-match XML Element . . . . .	45
10.6.	CARDDAV:limit XML Element . . . . .	46
10.6.1.	CARDDAV:nresults XML Element . . . . .	46
10.7.	CARDDAV:addressbook-multiget XML Element . . . . .	47
11.	Service Discovery via SRV Records . . . . .	47
12.	Internationalization Considerations . . . . .	48
13.	Security Considerations . . . . .	48
14.	IANA Consideration . . . . .	49
14.1.	Namespace Registration . . . . .	49
15.	Acknowledgments . . . . .	49
16.	References . . . . .	49
16.1.	Normative References . . . . .	49
16.2.	Informative References . . . . .	51
Appendix A.	Change History (to be removed prior to publication as an RFC) . . . . .	51
Author's Address	. . . . .	56

## 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) [RFC4510], Internet Message Support Protocol (IMSP) [IMSP] and Application Configuration Access Protocol (ACAP) [RFC2244], together with SyncML used for synchronization of such data.

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 as per WebDAV ACL [RFC3744].
3. Principal collections can be used to enumerate and query other users on the system as per WebDAV ACL [RFC3744].
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 WebDAV's use of XML.
6. Use of vCards [RFC2426] 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. Many of the alternative protocols also lack this ability. However, an extension for push notifications could easily be developed.

vCard 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.

An update to vCard - vCard v4 - is currently being developed [I-D.ietf-vcarddav-vcardrev] and is compatible with this specification.

## 2. 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].

This document uses XML DTD fragments ([W3C.REC-xml-20081126], Section 3.2) as a purely notational convention. WebDAV request and response bodies cannot be validated by a DTD due to the specific extensibility rules defined in Section 17 of [RFC4918] and due to the fact that all XML elements defined by this specification use the XML namespace name "DAV:". In particular:

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

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.

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 strings "DAV:" and

"CARDDAV:" will be prefixed to the element types, respectively.

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

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.

### 3. Requirements Overview

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

- o MUST support vCard v3 [RFC2426] as a media type for the address object resource format;
- o MUST support WebDAV Class 3 [RFC4918];
- o MUST support WebDAV ACL [RFC3744];
- o MUST support secure transport as defined in [RFC2818] using TLS [RFC5246] and using the certificate validation procedures described in [RFC5280];
- o MUST support ETags [RFC2616] with additional requirements specified in Section 6.3.2.3 of this document;
- o MUST support all address book reports defined in Section 8 of this document; and
- o MUST advertise support on all address book collections and address object resources for the address book reports in the DAV: supported-report-set property, as defined in Versioning Extensions to WebDAV [RFC3253].

In addition, a server:

- o SHOULD support vCard v4 [I-D.ietf-vcarddav-vcardrev] as a media type for the address object resource format;
- o SHOULD support the extended MKCOL method [RFC5689] to create address book collections as defined in Section 6.3.1 of this document.
- o SHOULD support the DAV:current-user-principal-URL property as defined in [RFC5397] to give clients a fast way to locate user

principals.

#### 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. Address data on the server can 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 locking 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.

#### 5.1.1. Data Type Conversion

Servers might support more than one primary media type for address object resources, for example vCard v3.0 and vCard v4.0. In such cases servers have to accept all media types that they advertise via the CARDDAV:supported-address-data WebDAV property (see Section 6.2.2).

However, clients can use standard HTTP content negotiation behavior (the Accept request header defined in Section 14.1 of [RFC2616]) to request that an address object resource's data be returned in a specific media type format. For example, a client only capable of handling vCard v3.0 would only want to have address object resources returned in v3.0 format.

Additionally, REPORT requests, defined later in this specification, allow for the return of address object resource data within an XML response body. Again, the client can use content negotiation to request that data be returned in a specific media type by specifying appropriate attributes on the CARDDAV:address-data XML element used in the request body (see Section 10.4).

In some cases it might not be possible for a server to convert from one media type to another. When that happens, the server **MUST** return the CARDDAV:supported-address-data-conversion precondition (see below) in the response body (when the failure to convert applies to the entire response) or use that same precondition code in the DAV:response XML element in the response for the targeted address object resource when one of the REPORTs defined below is used. See Section 8.7.2 for an example of this.



#### 5.1.1.1. Additional Precondition for GET

This specification creates additional preconditions for the GET method.

The new precondition is:

(CARDDAV:supported-address-data-conversion): The resource targeted by the GET request can be converted to the media type specified in the Accept request header included with the request;

#### 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 NOT 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. The xml:lang attribute can be used to add a language tag for the value of this property.

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 NOT 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-type+)>

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

Example:

```
<C:supported-address-data
  xmlns:C="urn:ietf:params:xml:ns:carddav">
  <C:address-data-type 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 in octets of a resource 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 NOT 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 decimal 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 [RFC5689] 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.

The DAV:displayname property can be used 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. 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 without overwriting a change made on the server, uses a specific ETag in an "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:

- o When a PUT operation of an address object resource into an address book collection occurs.
- o 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 a size in octets 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 vCard 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 NOT 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>/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 NOT 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 resource containing the corresponding contact information. The resource could be an address object resource in an address book collection, or it could be a resource in a "regular" collection.

Definition:

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

Example:

```
<C:principal-address xmlns:D="DAV:"  
  xmlns:C="urn:ietf:params:xml:ns:carddav">  
  <D:href>/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 NOT be returned by a PROPFIND DAV:allprop request.

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

Definition:

```
<!ELEMENT supported-collation-set (
  supported-collation
  supported-collation
  supported-collation*)>
<!-- Both "i;ascii-casemap" and "i;unicode-casemap"
will be present -->

<!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 vCard 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 vCard 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 vCard 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 CARDDAV: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 WebDAV property results. For instance, a request to retrieve the value of a WebDAV 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 MUST include a Depth header. The scope of the query is determined by the value of the Depth header. e.g., to query all address object resources in an address book collection, the REPORT would use the address book collection as the request-URI and specify a Depth of 1 or infinity.

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 vCard properties and parameters for which queries are supported by the server. i.e., if the CARDDAV:filter element attempts to reference an unsupported vCard 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. Limiting Results

A client can limit the number of results returned by the server through use of the CARDDAV:limit element in the request body. This is useful when clients are only interested in a few matches, or only have limited space to display results to users and thus don't need the overhead of receiving more than that. When the results are

truncated by the server, the server MUST follow the rules below for indicating a result set truncation to the client.

#### 8.6.2. Truncation of Results

A server MAY limit the number of resources in a response, for example, to limit the amount of work expended in processing a query, or as the result of an explicit limit set by the client. If the result set is truncated because of such a limit, the response MUST use status code 207, return a DAV:multistatus response body, and indicate a status of 507 (Insufficient Storage) for the request URI. That DAV:response element SHOULD include a DAV:error element with the DAV:number-of-matches-within-limits precondition, as defined in [RFC3744] (Section 9.2).

The server SHOULD also include the partial results in additional DAV:response elements. If a client requested limit is being applied, the 507 response for the request URI MUST NOT be included in calculating the limit (e.g., if the client requests that only a single result be returned, and multiple matches are present, then the DAV:multistatus response will include one DAV:response for the matching resource and one DAV:response for the 507 status on the request URI).

#### 8.6.3. Example: Partial Retrieval of vCards Matching 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>
```

#### 8.6.4. Example: Partial Retrieval of vCards Matching a Full Name or Email Address

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 or that contain an EMAIL property whose value contains other 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 test="anyof">
    <C:prop-filter name="FN">
      <C:text-match collation="i;unicode-casemap"
                    match-type="contains"
      >daboo</C:text-match>
    </C:prop-filter>
    <C:prop-filter name="EMAIL">
      <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:David Boo
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>
```

## 8.6.5. Example: Truncated Results

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 the DAV:getetag property for two results only. The server response includes a 507 status for the request URI indicating that there were more than two resources that matched the query, but that the server truncated the result set as requested by the client.

>> 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/>
  </D:prop>
  <C:filter test="anyof">
    <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:limit>
    <C:nresults>2</C:nresults>
  </C:limit>
</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/</D:href>

<D:status>HTTP/1.1 507 OK</D:status>

<D:error><D:number-of-matches-within-limits/></D:error>

<D:responsedescription xml:lang="en">

Only two matching records were returned

</D:responsedescription>

</D:response>

<D:response>

<D:href>/home/bernard/addressbook/v102.vcf</D:href>

<D:propstat>

<D:prop>

<D:getetag>"23ba4d-ff11fb"</D:getetag>

</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>

</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 10.7), which MUST contain at least one DAV:href XML element, and one optional CARDDAV:address-data element as defined in Section 10.4. If DAV:href elements are present, the scope of the request is the set of resources identified by these elements, which all need to be members (not necessarily internal members) of the resource identified by the Request-URI. Otherwise, the scope is the resource identified by the Request-URI itself.

The request MUST include a Depth: 0 header, however the actual scope of the REPORT is determined as described above.

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:

(CARDDAV: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 vCard 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
Depth: 1
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>
```

#### 8.7.2. Example: CARDDAV:addressbook-multiget Report

In this example, the client requests the server to return vCard v4.0 data 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/vcf3.vcf> exists but in a media type format that the server is unable to convert, resulting in an error status 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-multiget xmlns:D="DAV:"
                        xmlns:C="urn:ietf:params:xml:ns:carddav">
  <D:prop>
    <D:getetag/>
    <C:address-data type='text/vcard' version='4.0' />
  </D:prop>
  <D:href>/home/bernard/addressbook/vcf3.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/vcf3.vcf</D:href>
    <D:status>HTTP/1.1 415 Unsupported Media Type</D:status>
    <D:error><C:supported-address-data-conversion/></D:error>
    <D:responsedescription>Unable to convert from vCard v3.0
      to vCard v4.0</D:responsedescription>
  </D:response>
</D:multistatus>
```

## 9. Client 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. Avoiding Lost Updates

When resources are accessed by multiple clients, the possibility of clients overwriting each other's changes exists. To alleviate that, clients SHOULD use the If-Match request header on PUT requests with the ETag of the previously retrieved resource data to check whether the resource was modified since it was previously retrieved. If a pre-condition failure occurs, clients need to reload the resource and go through their own merge or conflict resolution process before writing back the data (again using the If-Match check).

### 9.3. Client Configuration

When CardDAV clients need to be configured, the key piece of information that they require is the principal-URL of the user whose address book information is desired. Servers SHOULD support the DAV:current-user-principal-URL property as defined in [RFC5397] to give clients a fast way to locate user principals.

Given support for SRV records (Section 11) and DAV:current-user-principal-URL [RFC5397], users only need enter a user identifier, host name and password to configure their client. The client would take the host name and do an SRV lookup to locate the CardDAV server, then execute an authenticated PROPFIND on the root / resource looking for the DAV:current-user-principal-URL property. The value returned gives the client direct access to the user's principal-URL and from there all the related CardDAV properties needed to locate address books.

### 9.4. Finding Other Users' Address Books

For address book sharing use cases, one might wish to find the address book belonging to another user. To find other users' address books on the same server, the DAV:principal-property-search REPORT [RFC3744] 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, limit?)>
```

#### 10.4. CARDDAV:address-data XML Element

Name: address-data

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

Purpose: Specifies one of the following:

1. The parts of an address object resource which should be returned by a given address book REPORT request, and the media type and version for the returned data;
2. The content of an address object resource in a response to an address book REPORT request.

Description: 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. Additionally a media type and version can be specified

to request that the server return the data in that format if possible.

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 CARDDAV:address-data XML element part of the address book REPORT request did not specify required vCard 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 &lt;, &gt;, &amp; 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 (allprop | prop*)>
```

when nested in the DAV:prop XML element in an address book 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 address book 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 each variant 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 vCard properties shall be returned.

Description: This element can be used when the client wants all vCard 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 vCard properties to return in the response.

Description: The "name" attribute specifies the name of the vCard 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.

vCard allows a "group" prefix to appear before a property name in the vCard data. When the "name" attribute does not specify a group prefix, it MUST match properties in the vCard data without a group prefix or with any group prefix. When the "name" attribute includes a group prefix, it MUST match properties that have exactly the same group prefix and name. e.g.: a "name" set to "TEL" will match "TEL", "X-ABC.TEL", "X-ABC-1.TEL" vCard properties. A "name" set to "X-ABC.TEL" will match an "X-ABC.TEL" vCard property only, it will not match "TEL" or "X-ABC-1.TEL".

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. The "test" attribute specifies whether any (logical OR) or all (logical AND) of the prop-filter tests needs to match in order for the overall filter to match.

Definition:

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

<!ATTLIST filter test (anyof | allof) "anyof">
<!-- test value:
      anyof logical OR for prop-filter matches
      allof logical AND for prop-filter matches -->
```

#### 10.5.1. CARDDAV:prop-filter XML Element

Name: prop-filter

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

Purpose: Limits the search to specific vCard 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 vCard property of the type specified by the "name" attribute exists, and the CARDDAV:prop-filter is empty, or it matches any specified CARDDAV:text-match or CARDDAV:param-filter conditions. The "test" attribute specifies whether any (logical OR) or all (logical AND) of the text-filter and param-filter tests need to match in order for the overall filter to match.

or:

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

vCard allows a "group" prefix to appear before a property name in the vCard data. When the "name" attribute does not specify a group prefix, it MUST match properties in the vCard data without a group prefix or with any group prefix. When the "name" attribute includes a group prefix, it MUST match properties that have

exactly the same group prefix and name. e.g.: a "name" set to "TEL" will match "TEL", "X-ABC.TEL", "X-ABC-1.TEL" vCard properties. A "name" set to "X-ABC.TEL" will match an "X-ABC.TEL" vCard property only, it will not match "TEL" or "X-ABC-1.TEL".

Definition:

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

<!ATTLIST prop-filter name CDATA #REQUIRED
                        test (anyof | allof) "anyof">
<!-- name value: a vCard property name (e.g., "NICKNAME")
test value:
    anyof logical OR for text-match/param-filter matches
    allof logical AND for text-match/param-filter matches -->
```

#### 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 vCard property or parameter does not exist.

Description: The CARDDAV:is-not-defined XML element specifies that a match occurs if the enclosing vCard 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 vCard property or parameter value.

Description: The CARDDAV:text-match XML element specifies text used for a substring match against the vCard 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:limit XML Element

Name: limit

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

Purpose: Specifies different types of limits that can be applied to the results returned by the server.

Description: The CARDDAV:limit XML element can be used to specify different types of limits that the client can request the server to apply to the results returned by the server. Currently only the CARDDAV:nresults limit can be used, other types of limit could be defined in the future.

Definition:

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

##### 10.6.1. CARDDAV:nresults XML Element

Name: nresults

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

Purpose: Specifies a limit on the number of results returned by the server.

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

Definition:

```
<!ELEMENT nresults (#PCDATA)>
<!-- nresults value: unsigned integer, must be digits -->
```

#### 10.7. 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. Service Discovery via SRV Records

[RFC2782] defines a DNS-based service discovery protocol that has been widely adopted as a means of locating particular services within a local area network and beyond, using SRV RR records.

This specification adds two service types for use with SRV records:

carddav: Identifies a CardDAV server that uses HTTP without TLS [RFC2818].

carddavs: Identifies a CardDAV server that uses HTTP with TLS [RFC2818].

Example: non-TLS service record

```
_carddav._tcp      SRV 0 1 80 addressbook.example.com.
```

Example: TLS service

```
_carddavs._tcp     SRV 0 1 443 addressbook.example.com.
```

## 12. 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).

## 13. 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 [RFC2617] 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. Clients **SHOULD** use TLS whenever possible.

With the ACL extension [RFC3744] 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. Servers are able to support address books that are "private" (accessible only to the "owner"), "shared" (accessible to the owner and other specified authenticated users), and "public" (accessible to any authenticated or unauthenticated users). When provisioning address books of a particular type, servers **MUST** ensure that the correct privileges are applied on creation, and in particular private and shared address books **MUST NOT** be accessible by unauthenticated users (to prevent data from being automatically searched or indexed by web "crawlers").

Clients **SHOULD** warn users in an appropriate fashion when they copy or move address data from a private address book to a shared address book or public address book. Clients **SHOULD** provide a clear



indication as to which address books are private, shared or public. Clients SHOULD provide an appropriate warning when changing access privileges for a private or shared address book with data so as to allow unauthenticated users access.

This specification currently relies on standard HTTP authentication mechanisms for identifying users. These comprise Basic and Digest authentication [RFC2617] as well as TLS [RFC2818] using client-side certificates.

#### 14. IANA Consideration

This document uses a URN to describe a new XML namespace conforming to the registry mechanism described in [RFC3688].

##### 14.1. 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 - not applicable for namespace registrations.

#### 15. 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: Mike Douglass, Stefan Eissing, Helge Hess, Arnaud Quillaud, Julian Reschke, Elias Sinderson, Greg Stein, Wilfredo Sanchez, and Simon Vaillancourt.

#### 16. References

##### 16.1. Normative References

[I-D.ietf-vcarddav-vcardrev]

Perreault, S. and P. Resnick, "vCard Format Specification", draft-ietf-vcarddav-vcardrev-08 (work in progress), July 2009.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Requirement Levels", BCP 14, RFC 2119, March 1997.

- [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.
- [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [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.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List

(CRL) Profile", RFC 5280, May 2008.

- [RFC5397] Sanchez, W. and C. Daboo, "WebDAV Current Principal Extension", RFC 5397, December 2008.
- [RFC5689] Daboo, C., "Extended MKCOL for Web Distributed Authoring and Versioning (WebDAV)", RFC 5689, September 2009.
- [W3C.REC-xml-20081126]  
Paoli, J., Yergeau, F., Bray, T., Sperberg-McQueen, C., and E. Maler, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008,  
<<http://www.w3.org/TR/2008/REC-xml-20081126>>.

## 16.2. 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.
- [RFC4510] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.

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

### Changes in -10

1. Updated to MKColExt RFC reference.
2. GenART Review: changed to clients SHOULD support TLS.
3. GenART Review: changed security considerations in relation to clients indicating which address books are private, shared or public.
4. IESG Review: re-wrote section on locking to instead describe how to avoid lost updates using ETags.
5. IESG Review: removed disadvantage describing stateless protocol nature.

6. IESG Review: clarified that 507 is only returned when truncation of the results set occurs.
7. IESG Review: added additional text in security considerations about the handling private, shared and public address books.
8. Fixed typos.
9. Fixed some XML example errors.

#### Changes in -09

1. AD Review: support for vCard v4 is now a SHOULD.
2. As a result of the above, added a sub-section on content conversion that defines a new precondition, and added an example of a conversion failure when doing a multiget.

#### Changes in -08

1. AD Review: added references to list in section 1.
2. AD Review: added reference to RFC5280 for cert validation procedures.
3. AD Review: added additional comment in addressbook-description property relating to use of xml:lang attribute.
4. AD Review: max-resource-size now explicitly stated to be a decimal integer.
5. AD Review: tweaked text for supported-collation-set to make it clear two will always be present.
6. AD Review: section title change to "Client Guidelines".
7. AD Review: finding address books section re-worded and reference added.
8. AD Review: re-worded prop-filter description to better explain that text-match and param-filter can be specified independently of each other.
9. AD Review: references added to security considerations.
10. AD Review: changed to RFC4510 reference.

#### Changes in -07

1. WGLC: changed all alprop behaviors to SHOULD NOT return in allprop PROPFIND.
2. WGLC: Reworked XML conventions section to come into line with text in extended MKCOL, and also updated W3C reference.
3. WGLC: Changed a couple of examples to use absolute path DAV:href values.
4. WGLC: Simplified IANA Considerations section.
5. WGLC: Added new Client Configuration section and removed reference to principal-match.
6. address-data element in supported-address-data changed to address-data-type.
7. REPORTs now require Depth to be present and the scope of matching resources is determined by the value of the Depth header.
8. Removed requirement that DAV:displayname should be unique at each level.

#### Changes in -06

1. WGLC: addressbook-home-set changed to SHOULD NOT return in allprop PROPFIND.
2. WGLC: principal-address description changed to note that the resource pointed to could be in a regular collection too.
3. Added new section describing how SRV and current-user-principal are used to bootstrap client configuration.
4. Removed discussion of using principal-match report.

#### Changes in -05

1. Removed mailing list discussion note from abstract.

#### Changes in -04

1. Tweaked limit element text to not imply any formal ordering of results.
2. Changed prop-filter element to allow zero or more text-match elements rather than zero or one.

3. Updated to RFC5397 reference.
4. Updated TLS reference to latest version RFC5246.
5. Boiler plate update.

#### Changes in -03

1. Added limit element to addressbook-query.
2. Specified how a server signals that query results have been truncated.
3. Minor stylistic changes.

#### Changes in -02

1. Added text to CARDDAV:prop and CARDDAV:prop-filter elements to explain how vCard "group" prefix on property names is handled.

#### Changes in -01

1. Added section on SRV records.

#### Changes in -00

1. Removed text describing other protocols.
2. Added comment about a new vcard spec being developed.
3. Added SHOULD support for the DAV:current-user-principal-URL property.
4. Added "anyof"/"allof" test attribute to query XML elements to support simple or/and combinations of tests.

#### Changes in pre-04

1. Renamed addressbook-data to address-data for consistency.
2. Fixed address-data element definition.

#### Changes in pre-03

1. Replaced MKADDRESSBOOK with extended MKCOL.
2. Now require i;unicode-casemap as a supported collation and make it the default.

3. No longer require i;octet as a supported 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 property to principal resources to allow a vcard to be associated with a principal.
9. XML definition clean-up.

#### Changes in pre-02

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 properties 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 to valid-address-data.

13. Now requires "i;unicasemao" collation.

Changes in pre-01

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](mailto:cyrus@daboo.name)  
URI: <http://www.apple.com/>





Network Working Group  
Internet-Draft  
Obsoletes: 2425, 2426, 4770  
(if approved)  
Updates: 2739 (if approved)  
Intended status: Standards Track  
Expires: November 27, 2011

S. Perreault  
Viagenie  
May 26, 2011

vCard Format Specification  
draft-ietf-vcarddav-vcardrev-22

## Abstract

This document defines the vCard data format for representing and exchanging a variety of information about individuals and other entities (e.g., formatted and structured name and delivery addresses, email address, multiple telephone numbers, photograph, logo, audio clips, etc.). This document obsoletes RFCs 2425, 2426, and 4770, and updates RFC 2739.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 27, 2011.

## Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

## Table of Contents

1. Introduction . . . . .	6
2. Conventions . . . . .	6
3. vCard Format Specification . . . . .	6
3.1. Charset . . . . .	6
3.2. Line Delimiting and Folding . . . . .	6
3.3. ABNF Format Definition . . . . .	7
3.4. Property Value Escaping . . . . .	10
4. Property Value Data Types . . . . .	10
4.1. TEXT . . . . .	12
4.2. URI . . . . .	13
4.3. DATE, TIME, DATE-TIME, DATE-AND-OR-TIME, and TIMESTAMP . . . . .	13
4.3.1. DATE . . . . .	13
4.3.2. TIME . . . . .	14
4.3.3. DATE-TIME . . . . .	14
4.3.4. DATE-AND-OR-TIME . . . . .	14
4.3.5. TIMESTAMP . . . . .	15
4.4. BOOLEAN . . . . .	15
4.5. INTEGER . . . . .	15
4.6. FLOAT . . . . .	16
4.7. UTC-OFFSET . . . . .	16
4.8. LANGUAGE-TAG . . . . .	16
5. Property Parameters . . . . .	16
5.1. LANGUAGE . . . . .	17
5.2. VALUE . . . . .	17
5.3. PREF . . . . .	18
5.4. ALTID . . . . .	18
5.5. PID . . . . .	20
5.6. TYPE . . . . .	20
5.7. MEDIATYPE . . . . .	21

5.8.	CALSCALE	21
5.9.	SORT-AS	22
5.10.	GEO	23
5.11.	TZ	24
6.	vCard Properties	24
6.1.	General Properties	24
6.1.1.	BEGIN	24
6.1.2.	END	25
6.1.3.	SOURCE	25
6.1.4.	KIND	26
6.1.5.	XML	28
6.2.	Identification Properties	29
6.2.1.	FN	29
6.2.2.	N	30
6.2.3.	NICKNAME	31
6.2.4.	PHOTO	31
6.2.5.	BDAY	32
6.2.6.	ANNIVERSARY	32
6.2.7.	GENDER	33
6.3.	Delivery Addressing Properties	34
6.3.1.	ADR	34
6.4.	Communications Properties	35
6.4.1.	TEL	35
6.4.2.	EMAIL	37
6.4.3.	IMPP	38
6.4.4.	LANG	38
6.5.	Geographical Properties	39
6.5.1.	TZ	39
6.5.2.	GEO	40
6.6.	Organizational Properties	40
6.6.1.	TITLE	40
6.6.2.	ROLE	41
6.6.3.	LOGO	42
6.6.4.	ORG	42
6.6.5.	MEMBER	43
6.6.6.	RELATED	44
6.7.	Explanatory Properties	45
6.7.1.	CATEGORIES	45
6.7.2.	NOTE	46
6.7.3.	PRODID	46
6.7.4.	REV	47
6.7.5.	SOUND	47
6.7.6.	UID	48
6.7.7.	CLIENTPIDMAP	48
6.7.8.	URL	49
6.7.9.	VERSION	50
6.8.	Security Properties	50
6.8.1.	KEY	50

6.9. Calendar Properties . . . . .	51
6.9.1. FBURL . . . . .	51
6.9.2. CALADRURI . . . . .	52
6.9.3. CALURI . . . . .	52
6.10. Extended Properties and Parameters . . . . .	53
7. Synchronization . . . . .	53
7.1. Mechanisms . . . . .	53
7.1.1. Matching vCard Instances . . . . .	53
7.1.2. Matching Property Instances . . . . .	54
7.1.3. PID Matching . . . . .	54
7.2. Example . . . . .	55
7.2.1. Creation . . . . .	55
7.2.2. Initial Sharing . . . . .	55
7.2.3. Adding and Sharing a Property . . . . .	56
7.2.4. Simultaneous Editing . . . . .	56
7.2.5. Global Context Simplification . . . . .	58
8. Example: Author's vCard . . . . .	59
9. Security Considerations . . . . .	59
10. IANA Considerations . . . . .	60
10.1. Media Type Registration . . . . .	60
10.2. Registering New vCard Elements . . . . .	61
10.2.1. Registration Procedure . . . . .	61
10.2.2. Vendor Namespace . . . . .	62
10.2.3. Registration Template for Properties . . . . .	63
10.2.4. Registration Template for Parameters . . . . .	63
10.2.5. Registration Template for Value Data Types . . . . .	64
10.2.6. Registration Template for Values . . . . .	64
10.3. Initial vCard Elements Registries . . . . .	65
10.3.1. Properties Registry . . . . .	65
10.3.2. Parameters Registry . . . . .	66
10.3.3. Value Data Types Registry . . . . .	67
10.3.4. Values Registries . . . . .	67
11. Acknowledgements . . . . .	70
12. References . . . . .	70
12.1. Normative References . . . . .	70
12.2. Informative References . . . . .	73
Appendix A. Differences from RFCs 2425 and 2426 . . . . .	75
A.1. New Structure . . . . .	75
A.2. Removed Features . . . . .	76
A.3. New Properties and Parameters . . . . .	76
A.4. Other Changes . . . . .	76
Appendix B. Change Log (to be removed by RFC Editor prior to publication) . . . . .	77
B.1. Changes in -22 . . . . .	77
B.2. Changes in -21 . . . . .	77
B.3. Changes in -20 . . . . .	77
B.4. Changes in -19 . . . . .	78
B.5. Changes in -18 . . . . .	78

B.6.	Changes in -17	. . . . .	78
B.7.	Changes in -16	. . . . .	79
B.8.	Changes in -15	. . . . .	79
B.9.	Changes in -14	. . . . .	80
B.10.	Changes in -13	. . . . .	81
B.11.	Changes in -12	. . . . .	81
B.12.	Changes in -11	. . . . .	82
B.13.	Changes in -10	. . . . .	83
B.14.	Changes in -09	. . . . .	83
B.15.	Changes in -08	. . . . .	83
B.16.	Changes in -07	. . . . .	84
B.17.	Changes in -06	. . . . .	84
B.18.	Changes in -05	. . . . .	85
B.19.	Changes in -04	. . . . .	85
B.20.	Changes in -03	. . . . .	86
B.21.	Changes in -02	. . . . .	86
B.22.	Changes in -01	. . . . .	87
B.23.	Changes in -00	. . . . .	87

## 1. Introduction

Electronic address books have become ubiquitous. Their increased presence on portable, connected devices as well as the diversity of platforms exchanging contact data call for a standard. This memo defines the vCard format, which allows the capture and exchange of information normally stored within an address book or directory application.

A high-level overview of the differences from RFCs 2425 and 2426 can be found in Appendix A.

## 2. Conventions

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

## 3. vCard Format Specification

The text/vcard MIME content type (hereafter known as "vCard", see Section 10.1) contains contact information, typically pertaining to a single contact or group of contacts. The content consists of one or more lines in the format given below.

### 3.1. Charset

The charset (see [RFC3536] for internationalization terminology) for vCard is UTF-8 as defined in [RFC3629]. There is no way to override this. It is invalid to specify a value other than "UTF-8" in the "charset" MIME parameter (see Section 10.1).

### 3.2. Line Delimiting and Folding

Individual lines within vCard are delimited by the [RFC5322] line break, which is a CRLF sequence (U+000D followed by U+000A). Long logical lines of text can be split into a multiple-physical-line representation using the following folding technique. Content lines SHOULD be folded to a maximum width of 75 octets, excluding the line break. Multi-octet characters MUST remain contiguous. The rationale for this folding process can be found in [RFC5322], Section 2.1.1.

A logical line MAY be continued on the next physical line anywhere between two characters by inserting a CRLF immediately followed by a single white space character (space (U+0020) or horizontal tab (U+0009)). The folded line MUST contain at least one character. Any sequence of CRLF followed immediately by a single white space

character is ignored (removed) when processing the content type. For example the line:

NOTE:This is a long description that exists on a long line.

can be represented as:

NOTE:This is a long description  
that exists on a long line.

It could also be represented as:

NOTE:This is a long descrip  
tion that exists o  
n a long line.

The process of moving from this folded multiple-line representation of a property definition to its single line representation is called unfolding. Unfolding is accomplished by regarding CRLF immediately followed by a white space character (namely HTAB (U+0009) or SPACE (U+0020)) as equivalent to no characters at all (i.e., the CRLF and single white space character are removed).

Note: It is possible for very simple implementations to generate improperly folded lines in the middle of a UTF-8 multi-octet sequence. For this reason, implementations SHOULD unfold lines in such a way as to properly restore the original sequence.

Note: Unfolding is done differently than in [RFC5322]. Unfolding in [RFC5322] only removes the CRLF, not the space following it.

Folding is done after any content encoding of a type value. Unfolding is done before any decoding of a type value in a content line.

### 3.3. ABNF Format Definition

The following ABNF uses the notation of [RFC5234], which also defines CRLF, WSP, DQUOTE, VCHAR, ALPHA, and DIGIT.

vcards-entity = 1\*vcards

vcards = "BEGIN:VCARD" CRLF  
"VERSION:4.0" CRLF  
1\*contentline  
"END:VCARD" CRLF

; A vCard object MUST include the VERSION and FN properties.  
; VERSION MUST come immediately after BEGIN:VCARD.



```

contentline = [group "."] name *("; " param) ":" value CRLF
; When parsing a content line, folded lines must first
; be unfolded according to the unfolding procedure
; described in section 3.2.
; When generating a content line, lines longer than 75
; characters SHOULD be folded according to the folding
; procedure described in section 3.2.

group = 1*(ALPHA / DIGIT / "-")
name = "SOURCE" / "KIND" / "FN" / "N" / "NICKNAME"
      / "PHOTO" / "BDAY" / "ANNIVERSARY" / "GENDER" / "ADR" / "TEL"
      / "EMAIL" / "IMPP" / "LANG" / "TZ" / "GEO" / "TITLE" / "ROLE"
      / "LOGO" / "ORG" / "MEMBER" / "RELATED" / "CATEGORIES"
      / "NOTE" / "PRODID" / "REV" / "SOUND" / "UID" / "CLIENTPIDMAP"
      / "URL" / "KEY" / "FBURL" / "CALADRURI" / "CALURI" / "XML"
      / iana-token / x-name
; Parsing of the param and value is based on the "name" as
; defined in ABNF sections below.
; Group and name are case-insensitive.

iana-token = 1*(ALPHA / DIGIT / "-")
; identifier registered with IANA

x-name = "x-" 1*(ALPHA / DIGIT / "-")
; Names that begin with "x-" or "X-" are
; reserved for experimental use, not intended for released
; products, or for use in bilateral agreements.

param = language-param / value-param / pref-param / pid-param
      / type-param / geo-parameter / tz-parameter / sort-as-param
      / calscale-param / any-param
; Allowed parameters depend on property name.

param-value = *SAFE-CHAR / DQUOTE *QSAFE-CHAR DQUOTE

any-param = (iana-token / x-name) "=" param-value *(", " param-value)

NON-ASCII = UTF8-2 / UTF8-3 / UTF8-4
; UTF8-{2,3,4} are defined in [RFC3629]

QSAFE-CHAR = WSP / "!" / %x23-7E / NON-ASCII
; Any character except CTLs, DQUOTE

SAFE-CHAR = WSP / "!" / %x23-39 / %x3C-7E / NON-ASCII
; Any character except CTLs, DQUOTE, ";", ":"

VALUE-CHAR = WSP / VCHAR / NON-ASCII
; Any textual character

```

A line that begins with a white space character is a continuation of the previous line, as described in Section 3.2. The white space character and immediately preceeding CRLF should be discarded when reconstructing the original line. Note that this line-folding convention differs from that found in [RFC5322], in that the sequence <CRLF><WSP> found anywhere in the content indicates a continued line and should be removed.

Property names and parameter names are case insensitive (e.g., the property name "fn" is the same as "FN" and "Fn"). Parameter values MAY be case-sensitive or case-insensitive, depending on their definition. Parameter values that are not explicitly defined as being case-sensitive are case-insensitive. Based on experience with vCard 3 interoperability, it is RECOMMENDED that property and parameter names be upper-case on output.

The group construct is used to group related properties together. The group name is a syntactic convention used to indicate that all property names prefaced with the same group name SHOULD be grouped together when displayed by an application. It has no other significance. Implementations that do not understand or support grouping MAY simply strip off any text before a "." to the left of the type name and present the types and values as normal.

Property cardinalities are indicated using the following notation, which is based on ABNF (see [RFC5234], section 3.6):

Cardinality	Meaning
1	Exactly one instance per vCard MUST be present.
*1	Exactly one instance per vCard MAY be present.
1*	One or more instances per vCard MUST be present.
*	One or more instances per vCard MAY be present.

Properties defined in a vCard instance may have multiple values depending on the property cardinality. The general rule for encoding multi-valued properties is to simply create a new content line for each value (including the property name). However, it should be noted that some value types support encoding multiple values in a single content line by separating the values with a comma ",". This approach has been taken for several of the content types defined below (date, time, integer, float).

### 3.4. Property Value Escaping

Some properties may contain one or more values delimited by a COMMA character (U+002C). Therefore, a COMMA character in a value MUST be escaped with a BACKSLASH character (U+005C), even for properties that don't allow multiple instances (for consistency).

Some properties (e.g. N and ADR) comprise multiple fields delimited by a SEMI-COLON character (U+003B). Therefore, a SEMI-COLON in a field of such a "compound" property MUST be escaped with a BACKSLASH character. SEMI-COLON characters in non-compound properties MAY be escaped. On input, an escaped SEMI-COLON character is never a field separator. An unescaped SEMI-COLON character may be a field separator, depending on the property in which it appears.

Furthermore, some fields of compound properties may contain a list of values delimited by a COMMA character. Therefore, a COMMA character in one of a field's values MUST be escaped with a BACKSLASH character, even for fields that don't allow multiple values (for consistency). Compound properties allowing multiple instances MUST NOT be encoded in a single content line.

Finally, BACKSLASH characters in values MUST be escaped with a BACKSLASH character. NEWLINE (U+000A) characters in values MUST be encoded by two characters: a BACKSLASH followed by either an 'n' (U+006E) or an 'N' (U+004E).

In all other cases, escaping MUST NOT be used.

### 4. Property Value Data Types

Standard value types are defined below.

```
value = text
      / text-list
      / date-list
      / time-list
      / date-time-list
      / date-and-or-time-list
      / timestamp-list
      / boolean
      / integer-list
      / float-list
      / URI ; from Section 3 of [RFC3986]
      / utc-offset
      / Language-Tag
      / iana-valuespec
      ; Actual value type depends on property name and VALUE parameter.
```

```

text = *TEXT-CHAR

TEXT-CHAR = "\\\" / "\",\" / "\\n\" / WSP / NON-ASCII
           / %x21-2B / %x2D-5B / %x5D-7E
           ; Backslashes, commas, and newlines must be encoded.

component = "\\\" / "\",\" / "\\;" / "\\n\" / WSP / NON-ASCII
           / %x21-2B / %x2D-3A / %x3C-5B / %x5D-7E
list-component = component *(",\" component)

text-list           = text           *(",\" text)
date-list           = date           *(",\" date)
time-list           = time           *(",\" time)
date-time-list      = date-time      *(",\" date-time)
date-and-or-time-list = date-and-or-time *(",\" date-and-or-time)
timestamp-list      = timestamp      *(",\" timestamp)
integer-list        = integer        *(",\" integer)
float-list          = float          *(",\" float)

boolean = "TRUE\" / "FALSE"
integer = [sign] 1*DIGIT
float   = [sign] 1*DIGIT [ "." 1*DIGIT]

sign = "+" / "-"

year   = 4DIGIT ; 0000-9999
month  = 2DIGIT ; 01-12
day    = 2DIGIT ; 01-28/29/30/31 depending on month and leap year
hour   = 2DIGIT ; 00-23
minute = 2DIGIT ; 00-59
second = 2DIGIT ; 00-58/59/60 depending on leap second
zone   = utc-designator / utc-offset
utc-designator = %x5A ; uppercase "Z"

date           = year [month day]
               / year "-" month
               / "---" month [day]
               / "---" "-" day
date-noreduc   = year month day
               / "---" month day
               / "---" "-" day
date-complete = year month day

time           = hour [minute [second]] [zone]
               / "-" minute [second] [zone]
               / "-" "-" second [zone]
time-notrunc   = hour [minute [second]] [zone]
time-complete  = hour minute second [zone]

```

```
time-designator = %x54 ; uppercase "T"
date-time = date-noreduc time-designator time-notrunc
timestamp = date-complete time-designator time-complete

date-and-or-time = date-time / date / time-designator time

utc-offset = sign hour [minute]

Language-Tag = <Language-Tag, defined in [RFC5646], Section 2.1>

iana-valuespec = <value-spec, see Section 12>
                  ; a publicly-defined valuetype format, registered
                  ; with IANA, as defined in section 12 of this
                  ; document
```

#### 4.1. TEXT

"text": The "text" value type should be used to identify values that contain human-readable text. As for the language, it is controlled by the LANGUAGE property parameter defined in Section 5.1.

Examples for "text":

```
this is a text value
this is one value,this is another
this is a single value\, with a comma encoded
```

A formatted text line break in a text value type MUST be represented as the character sequence backslash (U+005C) followed by a Latin small letter n (U+006E) or a Latin capital letter N (U+004E), that is "\n" or "\N".

For example a multiple line NOTE value of:

```
Mythical Manager
Hyjinx Software Division
BabsCo, Inc.
```

could be represented as:

```
NOTE:Mythical Manager\nHyjinx Software Division\n
      BabsCo\, Inc.\n
```

demonstrating the \n literal formatted line break technique, the CRLF-followed-by-space line folding technique, and the backslash escape technique.

#### 4.2. URI

"uri": The "uri" value type should be used to identify values that are referenced by a Uniform Resource Identifier (URI) instead of encoded in-line. These value references might be used if the value is too large, or otherwise undesirable to include directly. The format for the URI is as defined in Section 3 of [RFC3986]. Note that the value of a property of type "uri" is what the URI points to, not the URI itself.

Examples for "uri":

```
http://www.example.com/my/picture.jpg
ldap://ldap.example.com/cn=babs%20jensen
```

#### 4.3. DATE, TIME, DATE-TIME, DATE-AND-OR-TIME, and TIMESTAMP

"date", "time", "date-time", and "timestamp": Each of these value types is based on the definitions in [ISO.8601.2004] standard. Multiple such values can be specified using the comma-separated notation.

Only the basic format is supported.

##### 4.3.1. DATE

A calendar date as specified in [ISO.8601.2004] section 4.1.2.

Reduced accuracy, as specified in [ISO.8601.2004] sections 4.1.2.3 a) and b), but not c), is permitted.

Expanded representation, as specified in [ISO.8601.2004] section 4.1.4, is forbidden.

Truncated representation, as specified in [ISO.8601.2000] sections 5.2.1.3 d), e), and f), is permitted.

Examples for "date":

```
19850412
1985-04
1985
--0412
---12
```

Note the use of YYYY-MM in the second example above. YYYYMM is disallowed to prevent confusion with YYMMDD. Note also that YYYY-MM-DD is disallowed since we are using the basic format instead

of the extended format.

#### 4.3.2. TIME

A time of day as specified in [ISO.8601.2004] section 4.2.

Reduced accuracy, as specified in [ISO.8601.2004] section 4.2.2.3, is permitted.

Representation with decimal fraction, as specified in [ISO.8601.2004] section 4.2.2.4, is forbidden.

The midnight hour is always represented by 00, never 24 (see [ISO.8601.2004] section 4.2.3).

Truncated representation, as specified in [ISO.8601.2000] sections 5.3.1.4 a), b), and c), is permitted.

Examples for "time":

```
102200
1022
10
-2200
--00
102200Z
102200-0800
```

#### 4.3.3. DATE-TIME

A date and time of day combination as specified in [ISO.8601.2004] section 4.3.

Truncation of the date part, as specified in [ISO.8601.2000] section 5.4.2 c), is permitted.

Examples for "date-time":

```
19961022T140000
--1022T1400
---22T14
```

#### 4.3.4. DATE-AND-OR-TIME

Either a DATE-TIME, a DATE, or a TIME value. To allow unambiguous interpretation, a standalone TIME value is always preceded by a "T".

Examples for "date-and-or-time":

```
19961022T140000
--1022T1400
---22T14
19850412
1985-04
1985
--0412
---12
T102200
T1022
T10
T-2200
T--00
T102200Z
T102200-0800
```

#### 4.3.5. TIMESTAMP

A complete date and time of day combination as specified in [ISO.8601.2004] section 4.3.2.

Examples for "timestamp":

```
19961022T140000
19961022T140000Z
19961022T140000-05
19961022T140000-0500
```

#### 4.4. BOOLEAN

"boolean": The "boolean" value type is used to express boolean values. These values are case insensitive.

Examples:

```
TRUE
false
True
```

#### 4.5. INTEGER

"integer": The "integer" value type is used to express signed integers in decimal format. If sign is not specified, the value is assumed positive "+". Multiple "integer" values can be specified using the comma-separated notation. The maximum value is 9223372036854775807 and the minimum value is -9223372036854775808.



These limits correspond to a signed 64-bit integer using two's-complement arithmetic.

Examples:

```
1234567890
-1234556790
+1234556790,432109876
```

#### 4.6. FLOAT

"float": The "float" value type is used to express real numbers. If sign is not specified, the value is assumed positive "+". Multiple "float" values can be specified using the comma-separated notation. Implementations MUST support a precision equal or better than that of the IEEE "binary64" format [IEEE.754.2008].

Note: Scientific notation is disallowed. Implementors wishing to use their favorite language's %f formatting should be careful.

Examples:

```
20.30
1000000.0000001
1.333,3.14
```

#### 4.7. UTC-OFFSET

"utc-offset": The "utc-offset" value type specifies that the property value is a signed offset from UTC. This value type can be specified in the TZ property.

The value type is an offset from Coordinated Universal Time (UTC). It is specified as a positive or negative difference in units of hours and minutes (e.g., +hhmm). The time is specified as a 24-hour clock. Hour values are from 00 to 23, and minute values are from 00 to 59. Hour and minutes are 2-digits with high order zeroes required to maintain digit count. The basic format for ISO 8601 UTC offsets MUST be used.

#### 4.8. LANGUAGE-TAG

"language-tag": A single language tag, as defined in [RFC5646].

### 5. Property Parameters

A property can have attributes associated with it. These "property parameters" contain meta-information about the property or the

property value. In some cases the property parameter can be multi-valued in which case the property parameter value elements are separated by a COMMA (U+002C).

Property parameter value elements that contain the COLON (U+003A), SEMICOLON (U+003B) or COMMA (U+002C) character separators MUST be specified as quoted-string text values. Property parameter values MUST NOT contain the DQUOTE (U+0022) character. The DQUOTE character is used as a delimiter for parameter values that contain restricted characters or URI text.

Applications MUST ignore x-param and iana-param values they don't recognize.

#### 5.1. LANGUAGE

The LANGUAGE property parameter is used to identify data in multiple languages. There is no concept of "default" language, except as specified by any "Content-Language" MIME header parameter that is present [RFC3282]. The value of the LANGUAGE property parameter is a language tag as defined in Section 2 of [RFC5646].

Examples:

```
ROLE;LANGUAGE=tr:hoca
```

ABNF:

```
language-param = "LANGUAGE=" Language-Tag  
                ; Language-Tag is defined in section 2.1 of RFC 5646
```

#### 5.2. VALUE

The VALUE parameter is OPTIONAL, used to identify the value type (data type) and format of the value. The use of these predefined formats is encouraged even if the value parameter is not explicitly used. By defining a standard set of value types and their formats, existing parsing and processing code can be leveraged. The predefined data type values MUST NOT be repeated in COMMA separated value lists except within the N, NICKNAME, ADR, and CATEGORIES properties.

ABNF:

```
value-param = "VALUE=" value-type

value-type = "text"
            / "uri"
            / "date"
            / "time"
            / "date-time"
            / "date-and-or-time"
            / "timestamp"
            / "boolean"
            / "integer"
            / "float"
            / "utc-offset"
            / "language-tag"
            / iana-token ; registered as described in section 12
            / x-name
```

### 5.3. PREF

The PREF parameter is OPTIONAL and is used to indicate that the corresponding instance of a property is preferred by the vCard author. Its value MUST be an integer between 1 and 100 that quantifies the level of preference. Lower values correspond to a higher level of preference, 1 being most preferred.

When the parameter is absent, the default MUST be to interpret the property instance as being least preferred.

Note that the value of this parameter is to be interpreted only in relation to values assigned to other instances of the same property in the same vCard. A given value, or the absence of a value, MUST NOT be interpreted on its own.

This parameter MAY be applied to any property that allows multiple instances.

ABNF:

```
pref-param = "PREF=" (1*2DIGIT / "100")
              ; An integer between 1 and 100.
```

### 5.4. ALTID

The ALTID parameter is used to "tag" property instances as being alternative representations of the same logical property. For example, translations of a property in multiple languages generates

multiple property instances having different LANGUAGE (Section 5.1) parameter which are tagged with the same ALTID value.

This parameter's value is treated as an opaque string. Its sole purpose is to be compared for equality against other ALTID parameter values.

Two property instances are considered alternative representations of the same logical property if and only if their names as well as the value of their ALTID parameters are identical. Property instances without the ALTID parameter MUST NOT be considered an alternative representation of any other property instance. Values for the ALTID parameter are not globally unique: they MAY be reused for different property names.

Property instances having the same ALTID parameter value count as 1 toward cardinality. Therefore, since N (Section 6.2.2) has cardinality \*1 and TITLE (Section 6.6.1) has cardinality \*, these three examples would be legal:

```
N;ALTID=1;LANGUAGE=jp:<U+5C71><U+7530>;<U+592A><U+90CE>;;  
N;ALTID=1;LANGUAGE=en:Yamada;Taro;;;  
(<U+XXXX> denotes a UTF8-encoded Unicode character.)
```

```
TITLE;ALTID=1;LANGUAGE=fr:Patron  
TITLE;ALTID=1;LANGUAGE=en:Boss
```

```
TITLE;ALTID=1;LANGUAGE=fr:Patron  
TITLE;ALTID=1;LANGUAGE=en:Boss  
TITLE;ALTID=2;LANGUAGE=en:Chief vCard Evangelist
```

while this one would not:

```
N;ALTID=1;LANGUAGE=jp:<U+5C71><U+7530>;<U+592A><U+90CE>;;  
N:Yamada;Taro;;;  
(Two instances of the N property.)
```

and these three would be legal but questionable:

```
TITLE;ALTID=1;LANGUAGE=fr:Patron
TITLE;ALTID=2;LANGUAGE=en:Boss
(Should probably have the same ALTID value.)
```

```
TITLE;ALTID=1;LANGUAGE=fr:Patron
TITLE;LANGUAGE=en:Boss
(Second line should probably have ALTID=1.)
```

```
N;ALTID=1;LANGUAGE=jp:<U+5C71><U+7530>;<U+592A><U+90CE>;;
N;ALTID=1;LANGUAGE=en:Yamada;Taro;;
N;ALTID=1;LANGUAGE=en:Smith;John;;
(The last line should probably have ALTID=2. But that would be
illegal because N has cardinality *1.)
```

The ALTID property MAY also be used in many contexts other than with the LANGUAGE parameter. Here's an example with two representations of the same photo in different file formats:

```
PHOTO;ALTID=1:data:image/jpeg;base64,...
PHOTO;ALTID=1:data:image/jp2;base64,...
```

ABNF:

```
altid-param = "ALTID=" param-value
```

## 5.5. PID

The PID parameter is used to identify a specific property among multiple instances. It plays a role analogous to the UID property (Section 6.7.6) on a per-property instead of per-vCard basis. It MAY appear more than once in a given property. It MUST NOT appear on properties that may have only one instance per vCard. Its value is either a single small positive integer or a pair of small positive integers separated by a dot. Multiple values may be encoded in a single PID parameter by separating the values with a comma ",". See Section 7 for more details on its usage.

ABNF:

```
pid-param = "PID=" pid-value *("," pid-value)
pid-value = 1*DIGIT [ "." 1*DIGIT ]
```

## 5.6. TYPE

The TYPE parameter has multiple, different uses. In general, it is a way of specifying class characteristics of the associated property. Most of the time, its value is a comma-separated subset of a pre-defined enumeration. In this document, the following properties make

use of this parameter: FN, NICKNAME, PHOTO, ADR, TEL, EMAIL, IMPP, LANG, TZ, GEO, TITLE, ROLE, LOGO, ORG, RELATED, CATEGORIES, NOTE, SOUND, URL, KEY, FBURL, CALADRURI, and CALURI. The TYPE parameter MUST NOT be applied on other properties defined in this document.

The "work" and "home" values act like tags. The "work" value implies that the property is related to an individual's work place, while the "home" value implies that the property is related to an individual's personal life. When neither "work" nor "home" is present, it is implied that the property is related to both an individual's work place and personal life in the case that the KIND property's value is "individual", or to none in other cases.

ABNF:

```
type-param = "TYPE=" type-value *("," type-value)

type-value = "work" / "home" / type-param-tel
           / type-param-related / iana-token / x-name
           ; This is further defined in individual property sections.
```

#### 5.7. MEDIATYPE

The MEDIATYPE parameter is used with properties whose value is a URI. Its use is OPTIONAL. It provides a hint to the vCard consumer application about the media type [RFC2046] of the resource identified by the URI. Some URI schemes do not need this parameter. For example, the "data" scheme allows the media type to be explicitly indicated as part of the URI [RFC2397]. Another scheme, "http", provides the media type as part of the URI resolution process, with the Content-Type HTTP header [RFC2616]. The MEDIATYPE parameter is intended to be used with URI schemes that do not provide such functionality (e.g. "ftp" [RFC1738]).

ABNF:

```
mediatype-param = "MEDIATYPE=" mediatype
mediatype = type-name "/" subtype-name *( ";" attribute "=" value )
           ; "attribute" and "value" are from [RFC2045]
           ; "type-name" and "subtype-name" are from [RFC4288]
```

#### 5.8. CALSCALE

The CALSCALE parameter is identical to the CALSCALE property in iCalendar (see [RFC5545], section 3.7.1). It is used to define the calendar system in which a date or date-time value is expressed. The only value specified by iCalendar is "gregorian", which stands for the Gregorian system. It is the default when the parameter is

absent. Additional values may be defined in extension documents and registered from IANA (see Section 10.3.4). A vCard implementation MUST ignore properties with a CALSCALE parameter value that it does not understand.

ABNF:

```
calscale-param = "CALSCALE=" calscale-value
```

```
calscale-value = "gregorian" / iana-token / x-name
```

## 5.9. SORT-AS

The "sort-as" parameter is used to specify the string to be used for national-language-specific sorting. Without this information, sorting algorithms could incorrectly sort this vCard within a sequence of sorted vCards. When this property is present in a vCard, then the given strings are used for sorting the vCard.

This parameter's value is a comma-separated list which MUST have as many or fewer elements as the corresponding property value has components. This parameter's value is case-sensitive.

ABNF:

```
sort-as-param = "SORT-AS=" sort-as-value
```

```
sort-as-value = param-value *("," param-value)
```

Examples: For the case of surname and given name sorting, the following examples define common sort string usage with the N property.

FN:Rene van der Harten  
N;SORT-AS="Harten,Rene":van der Harten;Rene,J.;Sir;R.D.O.N.

FN:Robert Pau Shou Chang  
N;SORT-AS="Pau Shou Chang,Robert":Shou Chang;Robert,Pau;;

FN:Osamu Koura  
N;SORT-AS="Koura,Osamu":Koura;Osamu;;

FN:Oscar del Pozo  
N;SORT-AS="Pozo,Oscar":del Pozo Triscon;Oscar;;

FN:Chistine d'Aboville  
N;SORT-AS="Aboville,Christine":d'Aboville;Christine;;

FN:H. James de Mann  
N;SORT-AS="Mann,James":de Mann;Henry,James;;

If sorted by surname the results would be:

Christine d'Aboville  
Rene van der Harten  
Osamu Koura  
H. James de Mann  
Robert Pau Shou Chang  
Oscar del Pozo

If sorted by given name the results would be:

Christine d'Aboville  
H. James de Mann  
Osamu Koura  
Oscar del Pozo  
Rene van der Harten  
Robert Pau Shou Chang

#### 5.10. GEO

The GEO parameter can be used to indicate global positioning information that is specific to an address. Its value is the same as that of the GEO property (see Section 6.5.2).

ABNF:

geo-parameter = "GEO=" DQUOTE URI DQUOTE



### 5.11. TZ

The TZ parameter can be used to indicate time zone information that is specific to an address. Its value is the same as that of the TZ property.

ABNF:

```
tz-parameter = "TZ=" (param-value / DQUOTE URI DQUOTE)
```

## 6. vCard Properties

What follows is an enumeration of the standard vCard properties.

### 6.1. General Properties

#### 6.1.1. BEGIN

Purpose: To denote the beginning of a syntactic entity within a text/vcard content-type.

Value type: text

Cardinality: 1

Special notes: The content entity MUST begin with the BEGIN property with a value of "VCARD". The value is case-insensitive.

The BEGIN property is used in conjunction with the END property to delimit an entity containing a related set of properties within an text/vcard content-type. This construct can be used instead of including multiple vCards as body parts inside of a multipart/alternative MIME message. It is provided for applications that wish to define content that can contain multiple entities within the same text/vcard content-type or to define content that can be identifiable outside of a MIME environment.

ABNF:

```
BEGIN-param = 0" " ; no parameter allowed  
BEGIN-value = "VCARD"
```

Example:

```
BEGIN:VCARD
```

## 6.1.2. END

Purpose: To denote the end of a syntactic entity within a text/vcard content-type.

Value type: text

Cardinality: 1

Special notes: The content entity MUST end with the END type with a value of "VCARD". The value is case-insensitive.

The END property is used in conjunction with the BEGIN property to delimit an entity containing a related set of properties within an text/vcard content-type. This construct can be used instead of or in addition to wrapping separate sets of information inside additional MIME headers. It is provided for applications that wish to define content that can contain multiple entities within the same text/vcard content-type or to define content that can be identifiable outside of a MIME environment.

ABNF:

END-param = 0" " ; no parameter allowed  
END-value = "VCARD"

Example:

END:VCARD

## 6.1.3. SOURCE

Purpose: To identify the source of directory information contained in the content type.

Value type: uri

Cardinality: \*

Special notes: The SOURCE property is used to provide the means by which applications knowledgeable in the given directory service protocol can obtain additional or more up-to-date information from the directory service. It contains a URI as defined in [RFC3986] and/or other information referencing the vCard to which the information pertains. When directory information is available from more than one source, the sending entity can pick what it considers to be the best source, or multiple SOURCE properties can be included.

## ABNF:

```
SOURCE-param = "VALUE=uri" / pid-param / pref-param / altid-param  
               / mediatype-param / any-param  
SOURCE-value = URI
```

## Examples:

```
SOURCE:ldap://ldap.example.com/cn=Babs%20Jensen,%20o=Babsco,%20c=US
```

```
SOURCE:http://directory.example.com/addressbooks/jdoe/  
Jean%20Dupont.vcf
```

## 6.1.4. KIND

Purpose: To specify the kind of object the vCard represents.

Value type: A single text value.

Cardinality: \*1

Special notes: The value may be one of the following:

"individual" for a vCard representing a single person or entity.  
This is the default kind of vCard.

"group" for a vCard representing a group of persons or entities.  
The group's member entities can be other vCards or other types  
of entities, such as email addresses or web sites. A group  
vCard will usually contain MEMBER properties to specify the  
members of the group, but it is not required to. A group vCard  
without MEMBER properties can be considered an abstract  
grouping, or one whose members are known empirically (perhaps  
"IETF Participants" or "Republican U.S. Senators").

All properties in a group vCard apply to the group as a whole,  
and not to any particular MEMBER. For example, an EMAIL  
property might specify the address of a mailing list associated  
with the group, and an IMPP property might refer to a group  
chat room.

"org" for a vCard representing an organization. An organization  
vCard will not (in fact, MUST NOT) contain MEMBER properties,  
and so these are something of a cross between "individual" and  
"group". An organization is a single entity, but not a person.  
It might represent a business or government, a department or  
division within a business or government, a club, an  
association, or the like.

All properties in an organization vCard apply to the organization as a whole, as is the case with a group vCard. For example, an EMAIL property might specify the address of a contact point for the organization.

"location" for a named geographical place. A location vCard will usually contain a GEO property, but it is not required to. A location vCard without a GEO property can be considered an abstract location, or one whose definition is known empirically (perhaps "New England" or "The Seashore").

All properties in a location vCard apply to the location itself, and not with any entity that might exist at that location. For example, in a vCard for an office building, an ADR property might give the mailing address for the building, and a TEL property might specify the telephone number of the receptionist.

An x-name. vCards MAY include private or experimental values for KIND. Remember that x-name values are not intended for general use, and are unlikely to interoperate.

An iana-token. Additional values may be registered with IANA (see Section 10.3.4). A new value's specification document MUST specify which properties make sense for that new kind of vCard and which do not.

Implementations MUST support the specific string values defined above. If this property is absent, "individual" MUST be assumed as the default. If this property is present but the implementation does not understand its value (the value is an x-name or iana-token that the implementation does not support), the implementation SHOULD act in a neutral way, which usually means treating the vCard as though its kind were "individual". The presence of MEMBER properties MAY, however, be taken as an indication that the unknown kind is an extension of "group".

Clients often need to visually distinguish contacts based on what they represent, and the KIND property provides a direct way for them to do so. For example, when displaying contacts in a list, an icon could be displayed next to each one, using distinctive icons for the different kinds; a client might use an outline of a single person to represent an "individual", an outline of multiple people to represent a "group", and so on. Alternatively, or in addition, a client might choose to segregate different kinds of vCards to different panes, tabs, or selections in the user interface.

Some clients might also make functional distinctions among the kinds, ignoring "location" vCards for some purposes and considering only "location" vCards for others.

When designing those sorts of visual and functional distinctions, client implementations have to decide how to fit unsupported kinds into the scheme. What icon is used for them? The one for "individual"? A unique one, such as an icon of a question-mark? Which tab do they go into? It is beyond the scope of this specification to answer these questions, but these are things implementors need to consider.

ABNF:

```
KIND-param = "VALUE=text" / any-param
KIND-value  = "individual" / "group" / "org" / "location"
              / iana-token / x-name
```

Example:

This represents someone named Jane Doe working in the marketing department of the North American division of ABC Inc.

```
BEGIN:VCARD
VERSION:4.0
KIND:individual
FN:Jane Doe
ORG:ABC\, Inc.;North American Division;Marketing
END:VCARD
```

This represents the department itself, commonly known as ABC Marketing.

```
BEGIN:VCARD
VERSION:4.0
KIND:org
FN:ABC Marketing
ORG:ABC\, Inc.;North American Division;Marketing
END:VCARD
```

#### 6.1.5. XML

Purpose: To include extended XML-encoded vCard data in a plain vCard.

Value type: A single text value.

Cardinality: \*

Special notes: The content of this property is a single XML 1.0 [W3C.REC-xml-20081126] element whose namespace MUST be explicitly specified using the xmlns attribute and MUST NOT be the vCard 4 namespace ("urn:ietf:params:xml:ns:vcard-4.0"). (This implies that it cannot duplicate a standard vCard property.) The element is to be interpreted as if it was contained in a <vcard> element, as defined in [I-D.ietf-vcarddav-vcardxml].

The fragment is subject to normal line folding and escaping, i.e. replace all backslashes with "\\ ", then replace all newlines with "\\n", then fold long lines.

Support for this property is OPTIONAL, but implementations of this specification MUST preserve instances of this property when propagating vCards.

See [I-D.ietf-vcarddav-vcardxml] for more information on the intended use of this property.

ABNF:

XML-param = "VALUE=text" / altid-param

XML-value = text

## 6.2. Identification Properties

These types are used to capture information associated with the identification and naming of the entity associated with the vCard.

### 6.2.1. FN

Purpose: To specify the formatted text corresponding to the name of the object the vCard represents.

Value type: A single text value.

Cardinality: 1\*

Special notes: This property is based on the semantics of the X.520 Common Name attribute. The property MUST be present in the vCard object.

## ABNF:

```
FN-param = "VALUE=text" / type-param / language-param / altid-param  
          / pid-param / pref-param / any-param  
FN-value = text
```

## Example:

```
FN:Mr. John Q. Public\, Esq.
```

## 6.2.2. N

Purpose: To specify the components of the name of the object the vCard represents.

Value type: A single structured text value. Each component can have multiple values.

Cardinality: \*1

Special note: The structured property value corresponds, in sequence, to the Family Names (also known as surnames), Given Names, Additional Names, Honorific Prefixes, and Honorific Suffixes. The text components are separated by the SEMI-COLON character (U+003B). Individual text components can include multiple text values separated by the COMMA character (U+002C). This property is based on the semantics of the X.520 individual name attributes. The property SHOULD be present in the vCard object when the name of the object the vCard represents follows the X.520 model.

The SORT-AS parameter MAY be applied to this property.

## ABNF:

```
N-param = "VALUE=text" / sort-as-param / language-param  
          / altid-param / any-param  
N-value = list-component 4("; " list-component)
```

## Examples:

```
N:Public;John;Quinlan;Mr.;Esq.
```

```
N:Stevenson;John;Philip,Paul;Dr.;Jr.,M.D.,A.C.P.
```

## 6.2.3. NICKNAME

Purpose: To specify the text corresponding to the nickname of the object the vCard represents.

Value type: One or more text values separated by a COMMA character (U+002C).

Cardinality: \*

Special note: The nickname is the descriptive name given instead of or in addition to the one belonging to a the object the vCard represents. It can also be used to specify a familiar form of a proper name specified by the FN or N properties.

ABNF:

```
NICKNAME-param = "VALUE=text" / type-param / language-param
                  / altid-param / pid-param / pref-param / any-param
NICKNAME-value = text-list
```

Examples:

```
NICKNAME:Robbie
```

```
NICKNAME:Jim,Jimmie
```

```
NICKNAME;TYPE=work:Boss
```

## 6.2.4. PHOTO

Purpose: To specify an image or photograph information that annotates some aspect of the object the vCard represents.

Value type: A single URI.

Cardinality: \*

ABNF:

```
PHOTO-param = "VALUE=uri" / altid-param / type-param
               / mediatype-param / pref-param / pid-param / any-param
PHOTO-value = URI
```



## Examples:

PHOTO:http://www.example.com/pub/photos/jqpublic.gif

PHOTO:<...remainder of base64-encoded data...>

## 6.2.5. BDAY

Purpose: To specify the birth date of the object the vCard represents.

Value type: The default is a single date-and-or-time value. It can also be reset to a single text value.

Cardinality: \*1

## ABNF:

BDAY-param = BDAY-param-date / BDAY-param-text  
BDAY-value = date-and-or-time / text  
; Value and parameter MUST match.

BDAY-param-date = "VALUE=date-and-or-time"  
BDAY-param-text = "VALUE=text" / language-param

BDAY-param = / altid-param / calscale-param / any-param  
; calscale-param can only be present when BDAY-value is  
; date-and-or-time and actually contains a date or date-time.

## Examples:

BDAY:19960415  
BDAY:--0415  
BDAY;19531015T231000Z  
BDAY;VALUE=text:circa 1800

## 6.2.6. ANNIVERSARY

Purpose: The date of marriage, or equivalent, of the object the vCard represents.

Value type: The default is a single date-and-or-time value. It can also be reset to a single text value.

Cardinality: \*1

ABNF:

```
ANNIVERSARY-param = "VALUE=" ("date-and-or-time" / "text")
ANNIVERSARY-value = date-and-or-time / text
; Value and parameter MUST match.
```

```
ANNIVERSARY-param =/ altid-param / calscale-param / any-param
; calscale-param can only be present when ANNIVERSARY-value is
; date-and-or-time and actually contains a date or date-time.
```

Examples:

```
ANNIVERSARY:19960415
```

#### 6.2.7. GENDER

Purpose: To specify the components of the sex and gender identity of the object the vCard represents.

Value type: A single structured value with two components. Each component has a single text value.

Cardinality: \*1

Special notes: The components correspond, in sequence, to the sex (biological), and gender identity. Each component is optional.

Sex component: A single letter. M stands for "male", F stands for "female", O stands for "other", N stands for "none or not applicable", U stands for "unknown".

Gender identity component: Free-form text.

ABNF:

```
GENDER-param = "VALUE=text" / any-param
GENDER-value = sex [";" text]
```

```
sex = "" / "M" / "F" / "O" / "N" / "U"
```

Examples:

GENDER:M  
GENDER:F  
GENDER:M;Fellow  
GENDER:F;grrrl  
GENDER:O;intersex  
GENDER::it's complicated

### 6.3. Delivery Addressing Properties

These types are concerned with information related to the delivery addressing or label for the vCard object.

#### 6.3.1. ADR

Purpose: To specify the components of the delivery address for the vCard object.

Value type: A single structured text value, separated by the SEMI-COLON character (U+003B).

Cardinality: \*

Special notes: The structured type value consists of a sequence of address components. The component values MUST be specified in their corresponding position. The structured type value corresponds, in sequence, to

- the post office box;
- the extended address (e.g. apartment or suite number);
- the street address;
- the locality (e.g., city);
- the region (e.g., state or province);
- the postal code;
- the country name (full name in the language specified in Section 5.1).

When a component value is missing, the associated component separator MUST still be specified.

Experience with vCard 3 has shown that the first two components (post office box and extended address) are plagued with many interoperability issues. To ensure maximal interoperability, their values SHOULD be empty.

The text components are separated by the SEMI-COLON character (U+003B). Where it makes semantic sense, individual text components can include multiple text values (e.g., a "street" component with multiple lines) separated by the COMMA character (U+002C).

The property can include the "PREF" parameter to indicate the preferred delivery address when more than one address is specified.

The GEO and TZ parameters MAY be used with this property.

The property can also include a "LABEL" parameter to present a delivery delivery address label for the address. Its value is a plain-text string representing the formatted address. Newlines are encoded as \n, as they are for property values.

ABNF:

```
label-param = "LABEL=" param-value

ADR-param = "VALUE=text" / label-param / language-param
           / geo-parameter / tz-parameter / altid-param / pid-param
           / pref-param / type-param / any-param
ADR-value = ADR-component-pobox ";" ADR-component-ext ";"
           ADR-component-street ";" ADR-component-locality ";"
           ADR-component-region ";" ADR-component-code ";"
           ADR-component-country
ADR-component-pobox    = list-component
ADR-component-ext     = list-component
ADR-component-street   = list-component
ADR-component-locality = list-component
ADR-component-region   = list-component
ADR-component-code     = list-component
ADR-component-country  = list-component
```

Example: In this example the post office box and the extended address are absent.

```
ADR;GEO="geo:12.3457,78.910";LABEL="Mr. John Q. Public, Esq.\n
Mail Drop: TNE QB\n123 Main Street\nAny Town, CA 91921-1234\n
U.S.A.";;;123 Main Street;Any Town;CA;91921-1234;U.S.A.
```

## 6.4. Communications Properties

These properties describe information about how to communicate with the object the vCard represents.

### 6.4.1. TEL

**Purpose:** To specify the telephone number for telephony communication with the object the vCard represents.

**Value type:** By default it is a single free-form text value (for backward compatibility with vCard 3), but it SHOULD be reset to a URI value. It is expected that the URI scheme will be "tel", as specified in [RFC3966], but other schemes MAY be used.

**Cardinality:** \*

**Special notes:** This property is based on the X.520 Telephone Number attribute.

The property can include the "PREF" parameter to indicate a preferred-use telephone number.

The property can include the parameter "TYPE" to specify intended use for the telephone number. The predefined values for the TYPE parameter are:

Value	Description
text	Indicates that the telephone number supports text messages (SMS).
voice	Indicates a voice telephone number.
fax	Indicates a facsimile telephone number.
cell	Indicates a cellular or mobile telephone number.
video	Indicates a video conferencing telephone number.
pager	Indicates a paging device telephone number.
textphone	Indicates a telecommunication device for people with hearing or speech difficulties..

The default type is "voice". These type parameter values can be specified as a parameter list (e.g., TYPE=text;TYPE=voice) or as a value list (e.g., TYPE="text,voice"). The default can be overridden to another set of values by specifying one or more alternate values. For example, the default TYPE of "voice" can be reset to a VOICE and FAX telephone number by the value list TYPE="voice,fax".

If this property's value is a URI that can also be used for instant messaging, the IMPP (Section 6.4.3) property SHOULD be used in addition to this property.

## ABNF:

```
TEL-param = TEL-text-param / TEL-uri-param
TEL-value = TEL-text-value / TEL-uri-value
           ; Value and parameter MUST match

TEL-text-param = "VALUE=text"
TEL-text-value = text

TEL-uri-param = "VALUE=uri" / mediatype-param
TEL-uri-value = URI

TEL-param =/ type-param / pid-param / pref-param / altid-param
           / any-param

type-param-tel = "text" / "voice" / "fax" / "cell" / "video"
                / "pager" / "textphone" / iana-token / x-name
                ; type-param-tel MUST NOT be used with a property other than TEL.
```

## Example:

```
TEL;VALUE=uri;PREF=1;TYPE="voice,home":tel:+1-555-555-5555;ext=5555
TEL;VALUE=uri;TYPE=home:tel:+33-01-23-45-67
```

## 6.4.2. EMAIL

Purpose: To specify the electronic mail address for communication with the object the vCard represents.

Value type: A single text value.

Cardinality: \*

Special notes: The property can include the "PREF" parameter to indicate a preferred-use email address when more than one is specified.

Even though the value is free-form UTF-8 text, it is likely to be interpreted by an MUA as an "addr-spec", as defined in [RFC5322], section 3.4.1. Readers should also be aware of the current work toward internationalized email addresses [I-D.ietf-eai-rfc5335bis].

## ABNF:

```
EMAIL-param = "VALUE=text" / pid-param / pref-param / type-param
              / altid-param / any-param
EMAIL-value = text
```

Example:

```
EMAIL;TYPE=work:jpublic@xyz.example.com
```

```
EMAIL;PREF=1:jane_doe@example.com
```

#### 6.4.3. IMPP

Purpose: To specify the URI for instant messaging and presence protocol communications with the object the vCard represents.

Value type: A single URI.

Cardinality: \*

Special notes: The property may include the "PREF" parameter to indicate that this is a preferred address and has the same semantics as the "PREF" parameter in a TEL property.

If this property's value is a URI that can be used for voice and/or video, the TEL property (Section 6.4.1) SHOULD be used in addition to this property.

This property is adapted from [RFC4770], which is made obsolete by this document.

ABNF:

```
IMPP-param = "VALUE=uri" / pid-param / pref-param / type-param  
            / mediatype-param / altid-param / any-param  
IMPP-value = URI
```

Example:

```
IMPP;PREF=1:xmpp:alice@example.com
```

#### 6.4.4. LANG

Purpose: To specify the language(s) that may be used for contacting the entity associated with the vCard.

Value type: A single language-tag value.

Cardinality: \*

## ABNF:

```
LANG-param = "VALUE=language-tag" / pid-param / pref-param  
            / altid-param / type-param / any-param  
LANG-value = Language-Tag
```

## Example:

```
LANG;TYPE=work;PREF=1:en  
LANG;TYPE=work;PREF=2:fr  
LANG;TYPE=home:fr
```

## 6.5. Geographical Properties

These properties are concerned with information associated with geographical positions or regions associated with the object the vCard represents.

## 6.5.1. TZ

**Purpose:** To specify information related to the time zone of the object the vCard represents.

**Value type:** The default is a single text value. It can also be reset to a single URI or utc-offset value.

**Cardinality:** \*

**Special notes:** It is expected that names from the public-domain Olson database [TZ-DB] will be used, but this is not a restriction. See also [I-D.lear-iana-timezone-database].

Efforts are currently being directed at creating a standard URI scheme for expressing time zone information. Usage of such a scheme would ensure a high level of interoperability between implementations that support it.

Note that utc-offset values SHOULD NOT be used because the UTC offset varies with time - not just because of the usual daylight saving time shifts that occur in many regions, but often entire regions will "re-base" their overall offset. The actual offset may be +/- 1 hour (or perhaps a little more) than the one given.

## ABNF:



```
TZ-param = "VALUE=" ("text" / "uri" / "utc-offset")
TZ-value = text / URI / utc-offset
; Value and parameter MUST match

TZ-param =/ altid-param / pid-param / pref-param / type-param
           / mediatype-param / any-param
```

Examples:

```
TZ:Raleigh/North America
```

```
TZ;VALUE=utc-offset:-0500
; Note: utc-offset format is NOT RECOMMENDED.
```

#### 6.5.2. GEO

Purpose: To specify information related to the global positioning of the object the vCard represents.

Value type: A single URI.

Cardinality: \*

Special notes: The "geo" URI scheme [RFC5870] is particularly well-suited for this property, but other schemes MAY be used.

ABNF:

```
GEO-param = "VALUE=uri" / pid-param / pref-param / type-param
           / mediatype-param / altid-param / any-param
GEO-value = URI
```

Example:

```
GEO:geo:37.386013,-122.082932
```

### 6.6. Organizational Properties

These properties are concerned with information associated with characteristics of the organization or organizational units of the object that the vCard represents.

#### 6.6.1. TITLE

Purpose: To specify the position or job of the object the vCard represents.

Value type: A single text value.

Cardinality: \*

Special notes: This property is based on the X.520 Title attribute.

ABNF:

```
TITLE-param = "VALUE=text" / language-param / pid-param
              / pref-param / altid-param / type-param / any-param
TITLE-value = text
```

Example:

TITLE:Research Scientist

#### 6.6.2. ROLE

Purpose: To specify the function or part played in a particular situation by the object the vCard represents.

Value type: A single text value.

Cardinality: \*

Special notes: This property is based on the X.520 Business Category explanatory attribute. This property is included as an organizational type to avoid confusion with the semantics of the TITLE property and incorrect usage of that property when the semantics of this property is intended.

ABNF:

```
ROLE-param = "VALUE=text" / language-param / pid-param / pref-param
              / type-param / altid-param / any-param
ROLE-value = text
```

Example:

ROLE:Project Leader

## 6.6.3. LOGO

Purpose: To specify a graphic image of a logo associated with the object the vCard represents.

Value type: A single URI.

Cardinality: \*

ABNF:

```
LOGO-param = "VALUE=uri" / language-param / pid-param / pref-param
            / type-param / mediatype-param / altid-param / any-param
LOGO-value = URI
```

Examples:

LOGO:http://www.example.com/pub/logos/abccorp.jpg

LOGO:  
AQEEBQAwdzELMAkGA1UEBhMCVVMxLDAqBgNVBAoTIO5ldHNjYXB1IENvbW11bm  
1jYXRpb25zIENvcnBvcnF0aW9uMRwwGgYDVQQLEXNJbmZvcmlhdGlvbiBT  
<...the remainder of base64-encoded data...>

## 6.6.4. ORG

Purpose: To specify the organizational name and units associated with the vCard.

Value type: A single structured text value consisting of components separated by the SEMI-COLON character (U+003B).

Cardinality: \*

Special notes: The property is based on the X.520 Organization Name and Organization Unit attributes. The property value is a structured type consisting of the organization name, followed by zero or more levels of organizational unit names.

The SORT-AS parameter MAY be applied to this property.

ABNF:

```
ORG-param = "VALUE=text" / sort-as-param / language-param
            / pid-param / pref-param / altid-param / type-param
            / any-param
ORG-value = component *(";" component)
```

Example: A property value consisting of an organizational name, organizational unit #1 name and organizational unit #2 name.

ORG:ABC\, Inc.;North American Division;Marketing

#### 6.6.5. MEMBER

Purpose: To include a member in the group this vCard represents.

Value type: A single URI. It MAY refer to something other than a vCard object. For example, an e-mail distribution list could employ the "mailto" URI scheme [RFC6068] for efficiency.

Cardinality: \*

Special notes: This property MUST NOT be present unless the value of the KIND property is "group".

ABNF:

```
MEMBER-param = "VALUE=uri" / pid-param / pref-param / altid-param
               / mediatype-param / any-param
MEMBER-value = URI
```

Examples:

```
BEGIN:VCARD
VERSION:4.0
KIND:group
FN:The Doe family
MEMBER:urn:uuid:03a0e51f-d1aa-4385-8a53-e29025acd8af
MEMBER:urn:uuid:b8767877-b4a1-4c70-9acc-505d3819e519
END:VCARD
BEGIN:VCARD
VERSION:4.0
FN:John Doe
UID:urn:uuid:03a0e51f-d1aa-4385-8a53-e29025acd8af
END:VCARD
BEGIN:VCARD
VERSION:4.0
FN:Jane Doe
UID:urn:uuid:b8767877-b4a1-4c70-9acc-505d3819e519
END:VCARD
```

```
BEGIN:VCARD
VERSION:4.0
KIND:group
FN:Funky distribution list
MEMBER:mailto:subscriber1@example.com
MEMBER:xmpp:subscriber2@example.com
MEMBER:sip:subscriber3@example.com
MEMBER:tel:+1-418-555-5555
END:VCARD
```

#### 6.6.6. RELATED

Purpose: To specify a relationship between another entity and the entity represented by this vCard.

Value type: A single URI. It can also be reset to a single text value. The text value can be used to specify textual information.

Cardinality: \*

Special notes: The TYPE parameter MAY be used to characterize the related entity. It contains a comma-separated list of values that are registered from IANA as described in Section 10.2. The registry is pre-populated with the values defined in [xfn]. This document also specifies two additional values:

agent: an entity who may sometimes act on behalf of the entity associated with the vCard.

emergency: indicates an emergency contact

ABNF:

```

RELATED-param = RELATED-param-uri / RELATED-param-text
RELATED-value = URI / text
; Parameter and value MUST match.

RELATED-param-uri = "VALUE=uri" / mediatype-param
RELATED-param-text = "VALUE=text" / language-param

RELATED-param =/ pid-param / pref-param / altid-param / type-param
               / any-param

type-param-related = related-type-value *("," related-type-value)
; type-param-related MUST NOT be used with a property other than
; RELATED.

related-type-value = "contact" / "acquaintance" / "friend" / "met"
                   / "co-worker" / "colleague" / "co-resident"
                   / "neighbor" / "child" / "parent"
                   / "sibling" / "spouse" / "kin" / "muse"
                   / "crush" / "date" / "sweetheart" / "me"
                   / "agent" / "emergency"

```

#### Examples:

```

RELATED;TYPE=friend:urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
RELATED;TYPE=contact:http://example.com/directory/jdoe.vcf
RELATED;TYPE=co-worker;VALUE=text:Please contact my assistant Jane
Doe for any inquiries.

```

### 6.7. Explanatory Properties

These properties are concerned with additional explanations, such as that related to informational notes or revisions specific to the vCard.

#### 6.7.1. CATEGORIES

**Purpose:** To specify application category information about the vCard. Also known as "tags".

**Value type:** One or more text values separated by a COMMA character (U+002C).

**Cardinality:** \*

**ABNF:**

```

CATEGORIES-param = "VALUE=text" / pid-param / pref-param
                  / type-param / altid-param / any-param

```

CATEGORIES-value = text-list

Example:

CATEGORIES:TRAVEL AGENT

CATEGORIES:INTERNET,IETF,INDUSTRY,INFORMATION TECHNOLOGY

#### 6.7.2. NOTE

Purpose: To specify supplemental information or a comment that is associated with the vCard.

Value type: A single text value.

Cardinality: \*

Special notes: The property is based on the X.520 Description attribute.

ABNF:

NOTE-param = "VALUE=text" / language-param / pid-param / pref-param  
/ type-param / altid-param / any-param  
NOTE-value = text

Example:

NOTE:This fax number is operational 0800 to 1715  
EST\, Mon-Fri.

#### 6.7.3. PRODID

Purpose: To specify the identifier for the product that created the vCard object.

Type value: A single text value.

Cardinality: \*1

Special notes: Implementations SHOULD use a method such as that specified for Formal Public Identifiers in [ISO9070] or for Universal Resource Names in [RFC3406] to ensure that the text value is unique.

## ABNF:

```
PRODID-param = "VALUE=text" / any-param
PRODID-value = text
```

## Example:

```
PRODID:-//ONLINE DIRECTORY//NONSGML Version 1//EN
```

## 6.7.4. REV

Purpose: To specify revision information about the current vCard.

Value type: A single timestamp value.

Cardinality: \*1

Special notes: The value distinguishes the current revision of the information in this vCard for other renditions of the information.

## ABNF:

```
REV-param = "VALUE=timestamp" / any-param
REV-value = timestamp
```

## Example:

```
REV:19951031T222710Z
```

## 6.7.5. SOUND

Purpose: To specify a digital sound content information that annotates some aspect of the vCard. This property is often used to specify the proper pronunciation of the name property value of the vCard.

Value type: A single URI.

Cardinality: \*

## ABNF:

```
SOUND-param = "VALUE=uri" / language-param / pid-param / pref-param
              / type-param / mediatype-param / altid-param
              / any-param
SOUND-value = URI
```

## Example:



SOUND:CID:JOHNQPUBLIC.part8.19960229T080000.xyzMail@example.com

SOUND:data:audio/basic;base64,MIICajCCAdOgAwIBAgICBEUwDQYJKoZIh  
AQEEBQAwdzELMAkGA1UEBhMCVVMxLDAqBgNVBAoTI05ldHNjYXB1IENvbW11bm  
ljYXRpb25zIENvcnBvcnF0aW9uMRwwGgYDVQQLEXNJbmZvcmlhdGlvbiBTexN0  
<...the remainder of base64-encoded data...>

#### 6.7.6. UID

Purpose: To specify a value that represents a globally unique identifier corresponding to the entity associated with the vCard.

Value type: A single URI value. It MAY also be reset to free-form text.

Cardinality: \*1

Special notes: This property is used to uniquely identify the object that the vCard represents. The "uuid" URN namespace defined in [RFC4122] is particularly well-suited to this task, but other URI schemes MAY be used. Free-form text MAY also be used.

ABNF:

UID-param = UID-uri-param / UID-text-param  
UID-value = UID-uri-value / UID-text-value  
; Value and parameter MUST match.

UID-uri-param = "VALUE=uri"  
UID-uri-value = URI

UID-text-param = "VALUE=text"  
UID-text-value = text

UID-param =/ any-param

Example:

UID:urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6

#### 6.7.7. CLIENTPIDMAP

Purpose: To give a global meaning to a local PID source identifier.

Value type: A semicolon-separated pair of values. The first field is a small integer corresponding to the second field of a PID parameter instance. The second field is a URI. The "uuid" URN namespace defined in [RFC4122] is particularly well-suited to this

task, but other URI schemes MAY be used.

Cardinality: \*

Special notes: PID source identifiers (the source identifier is the second field in a PID parameter instance) are small integers that only have significance within the scope of a single vCard instance. Each distinct source identifier present in a vCard MUST have an associated CLIENTPIDMAP. See Section 7 for more details on the usage of CLIENTPIDMAP.

PID source identifiers MUST be strictly positive. Zero is not allowed.

As a special exception, the PID parameter MUST NOT be applied to this property.

ABNF:

```
CLIENTPIDMAP-param = any-param
CLIENTPIDMAP-value = 1*DIGIT ";" URI
```

Example:

```
TEL;PID=3.1,4.2;VALUE=uri:tel:+1-555-555-5555
EMAIL;PID=4.1,5.2:jdoe@example.com
CLIENTPIDMAP:1;urn:uuid:3df403f4-5924-4bb7-b077-3c711d9eb34b
CLIENTPIDMAP:2;urn:uuid:d89c9c7a-2e1b-4832-82de-7e992d95faa5
```

#### 6.7.8. URL

Purpose: To specify a uniform resource locator associated with the object that the vCard refers to. Examples for individuals include personal web sites, blogs, and social networking site identifiers.

Cardinality: \*

Value type: A single uri value.

ABNF:

```
URL-param = "VALUE=uri" / pid-param / pref-param / type-param
           / mediatype-param / altid-param / any-param
URL-value = URI
```

Example:

```
URL:http://example.org/restaurant.french/~chezchic.html
```

#### 6.7.9. VERSION

Purpose: To specify the version of the vCard specification used to format this vCard.

Value type: A single text value.

Cardinality: 1

Special notes: This property MUST be present in the vCard object, and it must appear immediately after BEGIN:VCARD. The value MUST be "4.0" if the vCard corresponds to this specification. Note that earlier versions of vCard allowed this property to be placed anywhere in the vCard object, or even to be absent.

ABNF:

```
VERSION-param = "VALUE=text" / any-param  
VERSION-value = "4.0"
```

Example:

```
VERSION:4.0
```

#### 6.8. Security Properties

These properties are concerned with the security of communication pathways or access to the vCard.

##### 6.8.1. KEY

Purpose: To specify a public key or authentication certificate associated with the object that the vCard represents.

Value type: A single URI. It can also be reset to a text value.

Cardinality: \*

ABNF:

KEY-param = KEY-uri-param / KEY-text-param  
 KEY-value = KEY-uri-value / KEY-text-value  
 ; Value and parameter MUST match.

KEY-uri-param = "VALUE=uri" / mediatype-param  
 KEY-uri-value = URI

KEY-text-param = "VALUE=text"  
 KEY-text-value = text

KEY-param =/ altid-param / pid-param / pref-param / type-param  
 / any-param

#### Examples:

KEY:http://www.example.com/keys/jdoe.cer

KEY;MEDIATYPE=application/pgp-keys:ftp://example.com/keys/jdoe

KEY:data:application/pgp-keys;base64,MIICajCCAdOgAwIBAgICBE  
 UwDQYJKoZIhvcNAQEEBQAwdzELMAkGA1UEBhMCVVMxLDAqBgNVBAoTI05l  
 <... remainder of base64-encoded data ...>

## 6.9. Calendar Properties

These properties are further specified in [RFC2739].

### 6.9.1. FBURL

Purpose: To specify the URI for the busy time associated with the object that the vCard represents.

Value type: A single URI value.

Cardinality: \*

Special notes: Where multiple FBURL properties are specified, the default FBURL property is indicated with the PREF parameter. The FTP [RFC1738] or HTTP [RFC2616] type of URI points to an iCalendar [RFC5545] object associated with a snapshot of the next few weeks or months of busy time data. If the iCalendar object is represented as a file or document, its file extension should be ".ifb".

#### ABNF:

FBURL-param = "VALUE=uri" / pid-param / pref-param / type-param  
 / mediatype-param / altid-param / any-param

FBURL-value = URI

Examples:

```
FBURL;PREF=1:http://www.example.com/busy/janedoe
FBURL;MEDIATYPE=text/calendar:ftp://example.com/busy/project-a.ifb
```

#### 6.9.2. CALADRURI

Purpose: To specify the calendar user address [RFC5545] to which a scheduling request [RFC5546] should be sent for the object represented by the vCard.

Value type: A single URI value.

Cardinality: \*

Special notes: Where multiple CALADRURI properties are specified, the default CALADRURI property is indicated with the PREF parameter.

ABNF:

```
CALADRURI-param = "VALUE=uri" / pid-param / pref-param / type-param
                  / mediatype-param / altid-param / any-param
CALADRURI-value = URI
```

Example:

```
CALADRURI;PREF=1:mailto:janedoe@example.com
CALADRURI:http://example.com/calendar/jdoe
```

#### 6.9.3. CALURI

Purpose: To specify the URI for a calendar associated with the object represented by the vCard.

Value type: A single URI value.

Cardinality: \*

Special notes: Where multiple CALURI properties are specified, the default CALURI property is indicated with the PREF parameter. The property should contain a URI pointing to an iCalendar [RFC5545] object associated with a snapshot of the user's calendar store. If the iCalendar object is represented as a file or document, its file extension should be ".ics".

## ABNF:

```
CALURI-param = "VALUE=uri" / pid-param / pref-param / type-param  
              / mediatype-param / altid-param / any-param  
CALURI-value = URI
```

## Examples:

```
CALURI;PREF=1:http://cal.example.com/calA  
CALURI;MEDIATYPE=text/calendar:ftp://ftp.example.com/calA.ics
```

## 6.10. Extended Properties and Parameters

The properties and parameters defined by this document can be extended. Non-standard, private properties and parameters with a name starting with "X-" may be defined bilaterally between two cooperating agents without outside registration or standardization.

## 7. Synchronization

vCard data often needs to be synchronized between devices. In this context, synchronization is defined as the intelligent merging of two representations of the same object. vCard 4.0 includes mechanisms to aid this process.

## 7.1. Mechanisms

Two mechanisms are available: the UID property is used to match multiple instances of the same vCard, while the PID parameter is used to match multiple instances of the same property.

The term "matching" is used here to mean recognizing that two instances are in fact representations of the same object. For example, a single vCard that is shared with someone results in two vCard instances. After they have evolved separately, they still represent the same object, and therefore may be matched by a synchronization engine.

## 7.1.1. Matching vCard Instances

vCard instances for which the UID properties (Section 6.7.6) are equivalent MUST be matched. Equivalence is determined as specified in [RFC3986], Section 6.

In all other cases, vCard instances MAY be matched at the discretion of the synchronization engine.

### 7.1.2. Matching Property Instances

Property instances belonging to unmatched vCards MUST NOT be matched.

Property instances whose name (e.g. EMAIL, TEL, etc.) is not the same MUST NOT be matched.

Property instances whose name is CLIENTPIDMAP are handled separately and MUST NOT be matched. The synchronization MUST ensure that there is consistency of CLIENTPIDMAPs among matched vCard instances.

Property instances belonging to matched vCards, whose name is the same, and whose maximum cardinality is 1 MUST be matched.

Property instances belonging to matched vCards, whose name is the same, and whose PID parameters match MUST be matched. See Section 7.1.3 for details on PID matching.

In all other cases, property instances MAY be matched at the discretion of the synchronization engine.

### 7.1.3. PID Matching

Two PID values for which the first fields are equivalent represent the same local value.

Two PID values representing the same local value and for which the second fields point to CLIENTPIDMAP properties whose second field URIs are equivalent (as specified in [RFC3986], Section 6) also represent the same global value.

PID parameters for which at least one pair of their values represent the same global value MUST be matched.

In all other cases, PID parameters MAY be matched at the discretion of the synchronization engine.

For example, PID value "5.1", in the first vCard below, and PID value "5.2", in the second vCard below, represent the same global value.

```
BEGIN:VCARD
VERSION:4.0
EMAIL;PID=4.2,5.1:jdoe@example.com
CLIENTPIDMAP:1;urn:uuid:3eef374e-7179-4196-a914-27358c3e6527
CLIENTPIDMAP:2;urn:uuid:42bcd5a7-1699-4514-87b4-056edf68e9cc
END:VCARD
```

```
BEGIN:VCARD
VERSION:4.0
EMAIL;PID=5.1,5.2:john@example.com
CLIENTPIDMAP:1;urn:uuid:0c75c629-6a8d-4d5e-a07f-1bb35846854d
CLIENTPIDMAP:2;urn:uuid:3eef374e-7179-4196-a914-27358c3e6527
END:VCARD
```

## 7.2. Example

### 7.2.1. Creation

The following simple vCard is first created on a given device.

```
BEGIN:VCARD
VERSION:4.0
UID:urn:uuid:4fbe8971-0bc3-424c-9c26-36c3eleff6b1
FN;PID=1.1:J. Doe
N:Doe;J.;;;
EMAIL;PID=1.1:jdoe@example.com
CLIENTPIDMAP:1;urn:uuid:53e374d9-337e-4727-8803-a1e9c14e0556
END:VCARD
```

This new vCard is assigned the UID "urn:uuid:4fbe8971-0bc3-424c-9c26-36c3eleff6b1" by the creating device. The FN and EMAIL properties are assigned the same local value of 1, and this value is given global context by associating it with "urn:uuid:53e374d9-337e-4727-8803-a1e9c14e0556", which represents the creating device. We are at liberty to reuse the same local value since instances of different properties will never be matched. The N property has no PID because it is forbidden by its maximum cardinality of 1.

### 7.2.2. Initial Sharing

This vCard is shared with a second device. Upon inspecting the UID property, the second device understands that this is a new vCard (i.e. unmatched) and thus the synchronization results in a simple copy.



### 7.2.3. Adding and Sharing a Property

A new phone number is created on the first device, then the vCard is shared with the second device. This is what the second device receives:

```
BEGIN:VCARD
VERSION:4.0
UID:urn:uuid:4fbe8971-0bc3-424c-9c26-36c3e1eff6b1
FN;PID=1.1:J. Doe
N:Doe;J.;.;
EMAIL;PID=1.1:jdoe@example.com
TEL;PID=1.1;VALUE=uri:tel:+1-555-555-5555
CLIENTPIDMAP:1;urn:uuid:53e374d9-337e-4727-8803-a1e9c14e0556
END:VCARD
```

Upon inspecting the UID property, the second device matches the vCard it received to the vCard that it already has stored. It then starts comparing the properties of the two vCards in same-named pairs.

The FN properties are matched because the PID parameters have the same global value. Since the property value is the same, no update takes place.

The N properties are matched automatically because their maximum cardinality is 1. Since the property value is the same, no update takes place.

The EMAIL properties are matched because the PID parameters have the same global value. Since the property value is the same, no update takes place.

The TEL property in the new vCard is not matched to any in the stored vCard because no property in the stored vCard has the same name. Therefore, this property is copied from the new vCard to the stored vCard.

The CLIENTPIDMAP property is handled separately by the synchronization engine. It ensures that it is consistent with the stored one. If it was not, the results would be up to the synchronization engine, and thus undefined by this document.

### 7.2.4. Simultaneous Editing

A new email address and a new phone number are added to the vCard on each of the two devices, and then a new synchronization event happens. Here are the vCards that are communicated to each other:

```
BEGIN:VCARD
VERSION:4.0
UID:urn:uuid:4fbe8971-0bc3-424c-9c26-36c3e1eff6b1
FN;PID=1.1:J. Doe
N:Doe;J.;.;
EMAIL;PID=1.1:jdoe@example.com
EMAIL;PID=2.1:boss@example.com
TEL;PID=1.1;VALUE=uri:tel:+1-555-555-5555
TEL;PID=2.1;VALUE=uri:tel:+1-666-666-6666
CLIENTPIDMAP:1;urn:uuid:53e374d9-337e-4727-8803-a1e9c14e0556
END:VCARD
```

```
BEGIN:VCARD
VERSION:4.0
UID:urn:uuid:4fbe8971-0bc3-424c-9c26-36c3e1eff6b1
FN;PID=1.1:J. Doe
N:Doe;J.;.;
EMAIL;PID=1.1:jdoe@example.com
EMAIL;PID=2.2:ceo@example.com
TEL;PID=1.1;VALUE=uri:tel:+1-555-555-5555
TEL;PID=2.2;VALUE=uri:tel:+1-666-666-6666
CLIENTPIDMAP:1;urn:uuid:53e374d9-337e-4727-8803-a1e9c14e0556
CLIENTPIDMAP:2;urn:uuid:1f762d2b-03c4-4a83-9a03-75ff658a6eee
END:VCARD
```

On the first device, the same PID source identifier (1) is reused for the new EMAIL and TEL properties. On the second device, a new source identifier (2) is generated, and a corresponding CLIENTPIDMAP property is created. It contains the second device's identifier, "urn:uuid:1f762d2b-03c4-4a83-9a03-75ff658a6eee".

The new EMAIL properties are unmatched on both sides since the PID global value is new in both cases. The sync thus results in a copy on both sides.

Although the situation appears to be the same for the TEL properties, in this case the synchronization engine is particularly smart and matches the two new TEL properties even though their PID global values are different. Note that in this case, the rules of section Section 7.1.2 state that two properties MAY be matched at the discretion of the synchronization engine. Therefore, the two properties are merged.

All this results in the following vCard which is stored on both devices:

```
BEGIN:VCARD
VERSION:4.0
UID:urn:uuid:4fbe8971-0bc3-424c-9c26-36c3e1eff6b1
FN:J. Doe
N:Doe;J.;.;
EMAIL;PID=1.1:jdoe@example.com
EMAIL;PID=2.1:boss@example.com
EMAIL;PID=2.2:ceo@example.com
TEL;PID=1.1;VALUE=uri:tel:+1-555-555-5555
TEL;PID=2.1,2.2;VALUE=uri:tel:+1-666-666-6666
CLIENTPIDMAP:1;urn:uuid:53e374d9-337e-4727-8803-a1e9c14e0556
CLIENTPIDMAP:2;urn:uuid:1f762d2b-03c4-4a83-9a03-75ff658a6eee
END:VCARD
```

#### 7.2.5. Global Context Simplification

The two devices finish their synchronization procedure by simplifying their global contexts. Since they haven't talked to any other device, the following vCard is for all purposes equivalent to the above. It is also shorter.

```
BEGIN:VCARD
VERSION:4.0
UID:urn:uuid:4fbe8971-0bc3-424c-9c26-36c3e1eff6b1
FN:J. Doe
N:Doe;J.;.;
EMAIL;PID=1.1:jdoe@example.com
EMAIL;PID=2.1:boss@example.com
EMAIL;PID=3.1:ceo@example.com
TEL;PID=1.1;VALUE=uri:tel:+1-555-555-5555
TEL;PID=2.1;VALUE=uri:tel:+1-666-666-6666
CLIENTPIDMAP:1;urn:uuid:53e374d9-337e-4727-8803-a1e9c14e0556
END:VCARD
```

The details of global context simplification are unspecified by this document. They are left up to the synchronization engine. This example is merely intended to illustrate the possibility, which investigating would be, in the authors' opinion, worthwhile.

## 8. Example: Author's vCard

```
BEGIN:VCARD
VERSION:4.0
FN:Simon Perreault
N:Perreault;Simon;;;ing. jr,M.Sc.
BDAY:--0203
ANNIVERSARY:20090808T1430-0500
GENDER:M
LANG;PREF=1:fr
LANG;PREF=2:en
ORG;TYPE=work:Viagenie
ADR;TYPE=work:;Suite D2-630;2875 Laurier;
  Quebec;QC;G1V 2M2;Canada
TEL;VALUE=uri;TYPE="work,voice";PREF=1:tel:+1-418-656-9254;ext=102
TEL;VALUE=uri;TYPE="work,cell,voice,video,text":tel:+1-418-262-6501
EMAIL;TYPE=work:simon.perreault@viagenie.ca
GEO;TYPE=work:geo:46.772673,-71.282945
KEY;TYPE=work;VALUE=uri:
  http://www.viagenie.ca/simon.perreault/simon.asc
TZ:-0500
URL;TYPE=home:http://nomis80.org
END:VCARD
```

## 9. Security Considerations

- o Internet mail is often used to transport vCards and is subject to many well known security attacks, including monitoring, replay, and forgery. Care should be taken by any directory service in allowing information to leave the scope of the service itself, where any access controls or confidentiality can no longer be guaranteed. Applications should also take care to display directory data in a "safe" environment
- o vCards can carry cryptographic keys or certificates, as described in Section 6.8.1.
- o vCards often carry information that can be sensitive (e.g. birthday, address, and phone information). Although vCards have no inherent authentication or confidentiality provisions, they can easily be carried by any security mechanism that transfers MIME objects to address authentication or confidentiality (e.g. S/MIME [RFC5751], OpenPGP [RFC4880]). In cases where the confidentiality or authenticity of information contained in vCard is a concern, the vCard SHOULD be transported using one of these secure mechanisms. The KEY property (Section 6.8.1) can be used to transport the public key used by these mechanisms.

- o The information in a vCard may become out of date. In cases where the vitality of data is important to an originator of a vCard, the SOURCE property (Section 6.1.3) SHOULD be specified. In addition, the "REV" type described in section Section 6.7.4 can be specified to indicate the last time that the vCard data was updated.
- o Many vCard properties may be used to transport URIs. Please refer to [RFC3986] section 7 for considerations related to URIs.

## 10. IANA Considerations

### 10.1. Media Type Registration

IANA is asked to register the following Media Type (in <<http://www.iana.org/assignments/media-types/>>), and to mark the text/directory Media Type as DEPRECATED.

To: [ietf-types@iana.org](mailto:ietf-types@iana.org)

Subject: Registration of media type text/vcard

Type name: text

Subtype name: vcard

Required parameters: none

Optional parameters: version

The "version" parameter is to be interpreted identically as the VERSION vCard property. If this parameter is present, all vCards in a text/vcard body part MUST have a VERSION property with value identical to that of this MIME parameter.

"charset": as defined for text/plain [RFC2046]; encodings other than UTF-8 [RFC3629] MUST NOT be used.

Encoding considerations: 8bit

Security considerations: See Section 9.

Interoperability considerations: The text/vcard media type is intended to identify vCard data of any version. There are older specifications of vCard [RFC2426][vCard21] still in common use. While these formats are similar, they are not strictly compatible. In general, it is necessary to inspect the value of the VERSION property (see Section 6.7.9) for identifying the standard to which a given vCard object conforms.

In addition, the following media types are known to have been used to refer to vCard data. They should be considered deprecated in favor of text/vcard.

- \* text/directory
- \* text/directory; profile=vcard
- \* text/x-vcard

Published specification: draft-ietf-vcarddav-vcardrev-22

Applications that use this media type: They are numerous, diverse, and include mail user agents, instant messaging clients, address book applications, directory servers, customer relationship management software.

Additional information:

Magic number(s):

File extension(s): .vcf .vcard

Macintosh file type code(s):

Person & email address to contact for further information: vCard discussion mailing list <vcarddav@ietf.org>

Intended usage: COMMON

Restrictions on usage: none

Author: Simon Perreault

Change controller: IETF

## 10.2. Registering New vCard Elements

This section defines the process for registering new or modified vCard elements (i.e. properties, parameters, value data types, and values) with IANA. It does not contain any immediate IANA actions.

### 10.2.1. Registration Procedure

The IETF will create a mailing list, vcarddav@ietf.org [1], which can be used for public discussion of vCard element proposals prior to registration. Use of the mailing list is strongly encouraged. The IESG will appoint a designated expert who will monitor the

vcarddav@ietf.org [1] mailing list and review registrations.

Registration of new vCard elements MUST be reviewed by the designated expert and published in an RFC. A Standard Track RFC is REQUIRED for the registration of new value data types that modify existing properties. A Standard Track RFC is also REQUIRED for registration of vCard elements that modify vCard elements previously documented in a Standard Track RFC.

The registration procedure begins when a completed registration template, defined in the sections below, is sent to vcarddav@ietf.org [1] and iana@iana.org [2]. The designated expert is expected to tell IANA and the submitter of the registration within two weeks whether the registration is approved, approved with minor changes, or rejected with cause. When a registration is rejected with cause, it can be re-submitted if the concerns listed in the cause are addressed. Decisions made by the designated expert can be appealed to the IESG Applications Area Director, then to the IESG. They follow the normal appeals procedure for IESG decisions.

Once the registration procedure concludes successfully, IANA creates or modifies the corresponding record in the vCard registry. The completed registration template is discarded.

An RFC specifying new vCard elements MUST include the completed registration templates, which MAY be expanded with additional information. These completed templates are intended to go in the body of the document, not in the IANA Considerations section.

Finally, note that there is an XML representation for vCard defined in [I-D.ietf-vcarddav-vcardxml]. An XML representation SHOULD be defined for new vCard elements.

#### 10.2.2. Vendor Namespace

The vendor namespace is used for vCard elements associated with commercially available products. "Vendor" or "producer" are construed as equivalent and very broadly in this context.

A registration may be placed in the vendor namespace by anyone who needs to interchange files associated with the particular product. However, the registration formally belongs to the vendor or organization handling the vCard elements in the namespace being registered. Changes to the specification will be made at their request, as discussed in subsequent sections.

vCard elements belonging to the vendor namespace will be distinguished by the "VND-" prefix. This is followed by an IANA-

registered Private Enterprise Number (PEN), a dash, and a vCard element designation of the vendor's choosing (e.g., "VND-123456-MUDPIE").

While public exposure and review of vCard elements to be registered in the vendor namespace is not required, using the vcarddav@ietf.org [1] mailing list for review is strongly encouraged to improve the quality of those specifications. Registrations in the vendor namespace may be submitted directly to the IANA.

#### 10.2.3. Registration Template for Properties

A property is defined by completing the following template.

Namespace: Empty for the global namespace, "VND-NNNN-" for a vendor-specific property (where NNNN is replaced by the vendor's PEN).

Property name: The name of the property.

Purpose: The purpose of the property. Give a short but clear description.

Value type: Any of the valid value types for the property value needs to be specified. The default value type also needs to be specified.

Cardinality: See Section 6.

Property parameters: Any of the valid property parameters for the property MUST be specified.

Description: Any special notes about the property, how it is to be used, etc.

Format definition: The ABNF for the property definition needs to be specified.

Example(s): One or more examples of instances of the property needs to be specified.

#### 10.2.4. Registration Template for Parameters

A parameter is defined by completing the following template.



Namespace: Empty for the global namespace, "VND-NNNN-" for a vendor-specific property (where NNNN is replaced by the vendor's PEN).

Parameter name: The name of the parameter.

Purpose: The purpose of the parameter. Give a short but clear description.

Description: Any special notes about the parameter, how it is to be used, etc.

Format definition: The ABNF for the parameter definition needs to be specified.

Example(s): One or more examples of instances of the parameter needs to be specified.

#### 10.2.5. Registration Template for Value Data Types

A value data type is defined by completing the following template.

Value name: The name of the value type.

Purpose: The purpose of the value type. Give a short but clear description.

Description: Any special notes about the value type, how it is to be used, etc.

Format definition: The ABNF for the value type definition needs to be specified.

Example(s): One or more examples of instances of the value type needs to be specified.

#### 10.2.6. Registration Template for Values

A value is defined by completing the following template.

Value: The value literal.

Purpose: The purpose of the value. Give a short but clear description.

Conformance: The vCard properties and/or parameters that can take this value needs to be specified.

Example(s): One or more examples of instances of the value needs to be specified.

The following is a fictitious example of a registration of a vCard value:

Value: supervisor

Purpose: It means that the related entity is the direct hierarchical superior (i.e. supervisor or manager) of the entity this vCard represents.

Conformance: This value can be used with the "TYPE" parameter applied on the "RELATED" property.

Example(s):

RELATED;TYPE=supervisor:urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6

### 10.3. Initial vCard Elements Registries

The IANA is requested to create and maintain the following registries for vCard elements with pointers to appropriate reference documents. The registries will be grouped together under the heading "vCard Elements".

#### 10.3.1. Properties Registry

The following table is to be used to initialize the properties registry.

Namespace	Property	Reference
	SOURCE	RFCXXXX, Section 6.1.3
	KIND	RFCXXXX, Section 6.1.4
	XML	RFCXXXX, Section 6.1.5
	FN	RFCXXXX, Section 6.2.1
	N	RFCXXXX, Section 6.2.2
	NICKNAME	RFCXXXX, Section 6.2.3
	PHOTO	RFCXXXX, Section 6.2.4
	BDAY	RFCXXXX, Section 6.2.5
	ANNIVERSARY	RFCXXXX, Section 6.2.6
	GENDER	RFCXXXX, Section 6.2.7
	ADR	RFCXXXX, Section 6.3.1
	TEL	RFCXXXX, Section 6.4.1
	EMAIL	RFCXXXX, Section 6.4.2
	IMPP	RFCXXXX, Section 6.4.3
	LANG	RFCXXXX, Section 6.4.4
	TZ	RFCXXXX, Section 6.5.1
	GEO	RFCXXXX, Section 6.5.2
	TITLE	RFCXXXX, Section 6.6.1
	ROLE	RFCXXXX, Section 6.6.2
	LOGO	RFCXXXX, Section 6.6.3
	ORG	RFCXXXX, Section 6.6.4
	MEMBER	RFCXXXX, Section 6.6.5
	RELATED	RFCXXXX, Section 6.6.6
	CATEGORIES	RFCXXXX, Section 6.7.1
	NOTE	RFCXXXX, Section 6.7.2
	PRODID	RFCXXXX, Section 6.7.3
	REV	RFCXXXX, Section 6.7.4
	SOUND	RFCXXXX, Section 6.7.5
	UID	RFCXXXX, Section 6.7.6
	CLIENTPIDMAP	RFCXXXX, Section 6.7.7
	URL	RFCXXXX, Section 6.7.8
	VERSION	RFCXXXX, Section 6.7.9
	KEY	RFCXXXX, Section 6.8.1
	FBURL	RFCXXXX, Section 6.9.1
	CALADRURI	RFCXXXX, Section 6.9.2
	CALURI	RFCXXXX, Section 6.9.3

### 10.3.2. Parameters Registry

The following table is to be used to initialize the parameters registry.

Namespace	Parameter	Reference
	LANGUAGE	RFCXXXX, Section 5.1
	VALUE	RFCXXXX, Section 5.2
	PREF	RFCXXXX, Section 5.3
	ALTID	RFCXXXX, Section 5.4
	PID	RFCXXXX, Section 5.5
	TYPE	RFCXXXX, Section 5.6
	MEDIATYPE	RFCXXXX, Section 5.7
	CALSCALE	RFCXXXX, Section 5.8
	SORT-AS	RFCXXXX, Section 5.9
	GEO	RFCXXXX, Section 5.10
	TZ	RFCXXXX, Section 5.11

#### 10.3.3. Value Data Types Registry

The following table is to be used to initialize the parameters registry.

Value Data Type	Reference
BOOLEAN	RFCXXXX, Section 4.4
DATE	RFCXXXX, Section 4.3.1
TIME	RFCXXXX, Section 4.3.2
DATE-TIME	RFCXXXX, Section 4.3.3
DATE-AND-OR-TIME	RFCXXXX, Section 4.3.4
TIMESTAMP	RFCXXXX, Section 4.3.5
FLOAT	RFCXXXX, Section 4.6
INTEGER	RFCXXXX, Section 4.5
TEXT	RFCXXXX, Section 4.1
URI	RFCXXXX, Section 4.2
UTC-OFFSET	RFCXXXX, Section 4.7
LANGUAGE-TAG	RFCXXXX, Section 4.8

#### 10.3.4. Values Registries

Separate tables will be used for property and parameter values.

The following table is to be used to initialize the property values registry.

Property	Value	Reference
BEGIN	VCARD	RFCXXXX, Section 6.1.1
END	VCARD	RFCXXXX, Section 6.1.2
KIND	individual	RFCXXXX, Section 6.1.4
KIND	group	RFCXXXX, Section 6.1.4
KIND	org	RFCXXXX, Section 6.1.4
KIND	location	RFCXXXX, Section 6.1.4

The following table is to be used to initialize the parameter values registry.

Property	Parameter	Value	Reference
FN, NICKNAME, PHOTO, ADR, TEL, EMAIL, IMPP, LANG, TZ, GEO, TITLE, ROLE, LOGO, ORG, RELATED, CATEGORIES, NOTE, SOUND, URL, KEY, FBURL, CALADRURI, and CALURI	TYPE	work	RFCXXXX, Section 5.6
FN, NICKNAME, PHOTO, ADR, TEL, EMAIL, IMPP, LANG, TZ, GEO, TITLE, ROLE, LOGO, ORG, RELATED, CATEGORIES, NOTE, SOUND, URL, KEY, FBURL, CALADRURI, and CALURI	TYPE	home	RFCXXXX, Section 5.6
TEL	TYPE	text	RFCXXXX, Section 6.4.1
TEL	TYPE	voice	RFCXXXX, Section 6.4.1
TEL	TYPE	fax	RFCXXXX, Section 6.4.1
TEL	TYPE	cell	RFCXXXX, Section 6.4.1
TEL	TYPE	video	RFCXXXX, Section 6.4.1
TEL	TYPE	pager	RFCXXXX, Section 6.4.1
BDAY, ANNIVERSARY	CALSCALE	gregorian	RFCXXXX, Section 6.2.5

RELATED	TYPE	contact	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	acquaintance	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	friend	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	met	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	co-worker	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	colleague	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	co-resident	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	neighbor	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	child	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	parent	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	sibling	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	spouse	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	kin	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	muse	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	crush	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	date	RFCXXXX, Section 6.6.6 and [xfn]

RELATED	TYPE	sweetheart	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	me	RFCXXXX, Section 6.6.6 and [xfn]
RELATED	TYPE	agent	RFCXXXX, Section 6.6.6
RELATED	TYPE	emergency	RFCXXXX, Section 6.6.6

## 11. Acknowledgements

The authors would like to thank Tim Howes, Mark Smith, and Frank Dawson, the original authors of [RFC2425] and [RFC2426], Pete Resnick, who got this effort started and provided help along the way, as well as the following individuals who have participated in the drafting, review and discussion of this memo:

Aki Niemi, Andy Mabbett, Alexander Mayrhofer, Alexey Melnikov, Anil Srivastava, Barry Leiba, Ben Fortuna, Bernard Desruisseaux, Bernie Hoeneisen, Bjoern Hoehrmann, Caleb Richarson, Chris Bryant, Chris Newman, Cyrus Daboo, Daisuke Miyakawa, Dan Brickley, Dan Mosedale, Dany Cauchie, Darryl Champagne, Dave Thewlis, Filip Navara, Florian Zeitz, Helge Hess, Jari Urpalainen, Javier Godoy, Jean-Luc Schellens, Joe Hildebrand, Jose Luis Gayosso, Joseph Smarr, Julian Reschke, Kepeng Li, Kevin Marks, Kevin Wu Won, Kurt Zeilenga. Lisa Dusseault, Marc Blanchet, Mark Paterson, Markus Lorenz, Michael Haardt, Mike Douglass, Nick Levinson, Peter K. Sheerin, Peter Mogensen, Peter Saint-Andre, Renato Iannella, Rohit Khare, Sly Gryphon, Stephane Bortzmeyer, Tantek Celik, and Zoltan Ordogh.

## 12. References

### 12.1. Normative References

- |                   |  |
|-------------------|--|
| [CCITT.E163.1988] | International Telephone and Telegraph Consultative Committee, "Numbering Plan for the International Telephone Service", CCITT Recommendation E.163, 1988.    |
| [CCITT.X121.1988] | International Telephone and Telegraph Consultative Committee, "International Numbering Plan for the Public Data Networks", CCITT Recommendation X.121, 1988. |

- [CCITT.X520.1988] International International Telephone and Telegraph Consultative Committee, "Information Technology - Open Systems Interconnection - The Directory: Selected Attribute Types", CCITT Recommendation X.520, November 1988.
- [CCITT.X521.1988] International International Telephone and Telegraph Consultative Committee, "Information Technology - Open Systems Interconnection - The Directory: Selected Object Classes", CCITT Recommendation X.521, November 1988.
- [I-D.ietf-vcarddav-vcardxml] Perreault, S., "vCard XML Representation", draft-ietf-vcarddav-vcardxml-08 (work in progress), April 2011.
- [IEEE.754.2008] Institute of Electrical and Electronics Engineers, "IEEE Standard for Floating-Point Arithmetic", IEEE Standard 754, August 2008.
- [ISO.8601.2000] International Organization for Standardization, "Data elements and interchange formats - Information interchange - Representation of dates and times", ISO Standard 8601, December 2000.
- [ISO.8601.2004] International Organization for Standardization, "Data elements and interchange formats - Information interchange - Representation of dates and times", ISO Standard 8601, December 2004.
- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format



- of Internet Message Bodies",  
RFC 2045, November 1996.
- [RFC2046] Freed, N. and N. Borenstein,  
"Multipurpose Internet Mail  
Extensions (MIME) Part Two: Media  
Types", RFC 2046, November 1996.
- [RFC2119] Bradner, S., "Key words for use in  
RFCs to Indicate Requirement  
Levels", BCP 14, RFC 2119,  
March 1997.
- [RFC2739] Small, T., Hennessy, D., and F.  
Dawson, "Calendar Attributes for  
vCard and LDAP", RFC 2739,  
January 2000.
- [RFC3629] Yergeau, F., "UTF-8, a  
transformation format of ISO  
10646", STD 63, RFC 3629,  
November 2003.
- [RFC3966] Schulzrinne, H., "The tel URI for  
Telephone Numbers", RFC 3966,  
December 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and  
L. Masinter, "Uniform Resource  
Identifier (URI): Generic Syntax",  
STD 66, RFC 3986, January 2005.
- [RFC4122] Leach, P., Mealling, M., and R.  
Salz, "A Universally Unique  
Identifier (UUID) URN Namespace",  
RFC 4122, July 2005.
- [RFC4288] Freed, N. and J. Klensin, "Media  
Type Specifications and  
Registration Procedures", BCP 13,  
RFC 4288, December 2005.
- [RFC5234] Crocker, D. and P. Overell,  
"Augmented BNF for Syntax  
Specifications: ABNF", STD 68,  
RFC 5234, January 2008.
- [RFC5322] Resnick, P., Ed., "Internet

Message Format", RFC 5322, October 2008.

[RFC5545] Desruisseaux, B., "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 5545, September 2009.

[RFC5546] Daboo, C., "iCalendar Transport-Independent Interoperability Protocol (iTIP)", RFC 5546, December 2009.

[RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

[RFC5870] Mayrhofer, A. and C. Spanring, "A Uniform Resource Identifier for Geographic Locations ('geo' URI)", RFC 5870, June 2010.

[W3C.REC-xml-20081126] Paoli, J., Yergeau, F., Bray, T., Maler, E., and C. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <<http://www.w3.org/TR/2008/REC-xml-20081126>>.

[xfn] Celik, T., Mullenweg, M., and E. Meyer, "XHTML Friends Network 1.1", <<http://gmpg.org/xfn/11>>.

## 12.2. Informative References

[I-D.ietf-eai-rfc5335bis] Yang, A. and S. Steele, "Internationalized Email Headers", draft-ietf-eai-rfc5335bis-10 (work in progress), March 2011.

[I-D.lear-iana-timezone-database] Lear, E. and P. Eggert, "IANA Procedures for Maintaining the Timezone Database", draft-lear-iana-timezone-database-04 (work in progress), May 2011.

- [ISO9070] The International Organization for Standardization, "ISO 9070, Information Processing - SGML support facilities - Registration Procedures for Public Text Owner Identifiers", April 1991.
- [RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.
- [RFC2397] Masinter, L., "The "data" URL scheme", RFC 2397, August 1998.
- [RFC2425] Howes, T., Smith, M., and F. Dawson, "A MIME Content-Type for Directory Information", RFC 2425, September 1998.
- [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.
- [RFC3282] Alvestrand, H., "Content Language Headers", RFC 3282, May 2002.
- [RFC3406] Daigle, L., van Gulik, D., Iannella, R., and P. Faltstrom, "Uniform Resource Names (URN) Namespace Definition Mechanisms", BCP 66, RFC 3406, October 2002.
- [RFC3536] Hoffman, P., "Terminology Used in Internationalization in the IETF", RFC 3536, May 2003.
- [RFC4770] Jennings, C. and J. Reschke, Ed., "vCard Extensions for Instant Messaging (IM)", RFC 4770, January 2007.

- [RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, November 2007.
- [RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.
- [RFC6068] Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", RFC 6068, October 2010.
- [TZ-DB] Olson, A., "Time zone code and data",  
<ftp://elsie.nci.nih.gov/pub/>.
- [vCard21] Internet Mail Consortium, "vCard - The Electronic Business Card Version 2.1", September September.

#### URIs

- [1] <mailto:vcardsdav@ietf.org>
- [2] <mailto:iana@iana.org>

#### Appendix A. Differences from RFCs 2425 and 2426

This appendix contains a high-level overview of the major changes that have been made in the vCard specification from RFCs 2425 and 2426. It is incomplete as it only lists the most important changes.

##### A.1. New Structure

- o [RFC2425] and [RFC2426] have been merged.
- o vCard is now not only a MIME type but a stand-alone format.
- o A proper MIME type registration form has been included.
- o UTF-8 is now the only possible character set.
- o New vCard elements can be registered from IANA.

#### A.2. Removed Features

- o The CONTEXT and CHARSET parameters are no more.
- o The NAME, MAILER, LABEL, and CLASS properties are no more.
- o The "intl", "dom", "postal", and "parcel" TYPE parameter values for the ADR property have been removed.
- o Inline vCards (such as the value of the AGENT property) are no longer supported.

#### A.3. New Properties and Parameters

- o The KIND, GENDER, LANG, ANNIVERSARY, XML, and CLIENTPIDMAP properties have been added.
- o [RFC2739], which defines the FBURL, CALADRURI, CAPURI, and CALURI properties, has been merged in.
- o [RFC4770], which defines the IMPP property, has been merged in.
- o The "work" and "home" TYPE parameter values are now applicable to many more properties.
- o The "pref" value of the TYPE parameter is now a parameter of its own, with a positive integer value indicating the level of preference.
- o The ALTID and PID parameters have been added.
- o The MEDIATYPE parameter has been added and replaces the TYPE parameter when it was used for indicating the media type of the property's content.

#### A.4. Other Changes

- o Synchronization is addressed in Section 7.
- o The N property is no longer mandatory.
- o In the N property, each component may now contain multiple comma-separated values.
- o The value of TEL is now a URI.
- o The AGENT property was replaced with a type of RELATED.

- o Date and time values now only support the basic format. Truncation is now supported.

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

### B.1. Changes in -22

- o Made reference to RFC3536 informative instead of normative.
- o Adjusted ABNF for readability (no normative changes).
- o Added informative reference to draft-lear-iana-timezone-database.
- o Tweaked Appendix A a little bit.
- o s/privacy/confidentiality/
- o Changed all references to ASCII characters into Unicode code points.
- o Added pointer to URI security considerations.
- o Adjusted GENDER examples.
- o Clearly ask IANA to mark text/directory as DEPRECATED.
- o Adjusted updating/obsoleting text in abstract.

### B.2. Changes in -21

- o Minor ABNF clarification.

### B.3. Changes in -20

- o Reworded security considerations.
- o Specified range of integer and float value types.
- o Adjusted IANA considerations wording.
- o Added missing date-and-or-type to the VALUE ABNF.
- o s/vcard@ietf.org/vcarddav@ietf.org/
- o More precise wording: charset instead of character set

- o Make parameter values case-insensitive by default.
- o Escaping newlines can be done with \n or \N.
- o Make SORT-AS value case-sensitive.
- o Better formatting for ADR special notes.
- o Removed Pete Resnick's example vCard.
- o vcarddav@ietf.org is the contact person for the media type.
- o Add missing pref-param, pid-param, and any-param to the PHOTO property.
- o Re-added missing section on UTC-OFFSET value type.

#### B.4. Changes in -19

- o Fixed nits.
- o Fixed MEDIATYPE ABNF.

#### B.5. Changes in -18

- o Fix missing text in KIND section.

#### B.6. Changes in -17

- o s/individual/entity/
- o Moved section 3.4 for better flow.
- o Removed text overriding general MIME and HTTP rules.
- o Fixed redundant ABNF.
- o Fixed parameter value quoting in examples.
- o Added informative reference for Content-Language header.
- o Moved cardinality table earlier for better flow.
- o Added normative reference for XML 1.0.
- o Added informative reference for mailto: scheme.

- o Clarified where the VERSION property must go.
- o Created the MEDIATYPE parameter.
- o Fixed text/vcard encoding considerations.
- o Added .vcard file extension.
- o Removed need for extension documents to contain tables in their IANA Considerations section.
- o Removed underspecified "Status" column in IANA registry.
- o Moved obsoleted references to Informative section.
- o Moved iCalendar references to Normative section.
- o Expanded specification of KIND values.
- o Recommend defining an XML representation for new vCard elements.

#### B.7. Changes in -16

- o RELATED: Added XFN values into IANA registry.
- o RELATED: Added values "agent" and "emergency".
- o EMAIL is now free-form text, with informative reference to EAI.
- o GENDER now has two components: one for biological sex, the other for gender identity.
- o Bugfixes to the core ABNF.
- o Clarified IANA considerations.
- o UID may be reset to free-form text.

#### B.8. Changes in -15

- o Reverted N to the 5-component format of vCard 3.
- o Removed DDAY, BIRTH, and DEATH.
- o First two components in ADR SHOULD be empty.
- o Removed the LABEL property.



- o Removed the binary value type and the ENCODING and FMTTYPE parameters.
- o Renamed SEX to GENDER. Set predefined values to "male" and "female".
- o Reverted TEL to take a text value by default, but SHOULD be reset to a URI.
- o Refer to iCalendar for CALSCALE.
- o Removed the "thing" value for KIND.
- o RELATED now uses XFN 1.1 for its value.
- o Dropped the VERSION parameter. XML MUST be version 1.0.
- o Dropped the CLASS property.
- o Property and parameter names SHOULD be upper-case.
- o Use ABNF for cardinality notation.

#### B.9. Changes in -14

- o DQUOTE is US-ASCII decimal 34, not 22.
- o Removed unused reference to RFC 2046.
- o Updated reference to draft-ietf-vcarddav-vcardxml.
- o Small fixes to the IANA registration text.
- o Added notes on the usage of TEL and IMPP properties.
- o Clarified "country name" component of ADR property.
- o Removed usage of undefined type value "msg" in TEL example.
- o Fixed parameter value quoting rules and ABNF.
- o Added example to LANGUAGE property.
- o Fixed synchronization example regarding the cardinality of FN.
- o Added implementation note for float value type.

- o Removed advice for always including VALUE parameter.
- o FMTTYPE MUST include the full MIME type.
- o Made ADR's ABNF more verbose.
- o Organized TEL TYPE values in a table.
- o Replaced TOP-SECRET example with NOINDEX.

#### B.10. Changes in -13

- o Changed global ABNF to make explicit that VERSION comes first.
- o Reworked example for LANGUAGE property.
- o s/TYPE/FMTTYPE/ in two examples.
- o Allow LANGUAGE parameter for text-valued BDAY, DDAY, and RELATED.
- o Tightened language on LANGUAGE parameter regarding cardinality.
- o Removed the NAME property.
- o Adjusted semi-colon escaping rules.
- o Added the ALTID parameter.

#### B.11. Changes in -12

- o Fixed example of LANGUAGE cardinality.
- o Added note about YYYY-MM date format.
- o Fixed appendix A.
- o VERSION property must come first.
- o Fixed mistake in PID example.
- o Removed two changes from Appendix A which were probably intended to go into this change log section.
- o Explicitly state that the value for the BEGIN and END properties is case-insensitive.
- o Removed the SORT-STRING property. Created the SORT-AS parameter.

- o "T" and "Z" in dates and times are now mandatory uppercase.
- o Added the "version" MIME parameter.
- o More consistent capitalization.
- o Added missing "ANNIVERSARY" in name production rule.
- o Added missing calscale-param in param production rule.
- o Moved definition of GEO and TZ parameters to section 5.
- o Simplified discussion of encoding.
- o Removed restriction for "work" and "home" values of TYPE parameter w.r.t. the KIND property.
- o XML value may now be binary.
- o Created VERSION parameter for XML.
- o BIRTH and DEATH can now have URI values.
- o Created the FMTTYPE parameter.
- o KEY can now take a text value.
- o Added references to RFC 5545 (iCalendar).
- o Added namespace column in parameters registry.

#### B.12. Changes in -11

- o Change "XML chunk" to "XML fragment".
- o Cite W3C document containing definition of "fragment".
- o Added "XML" to property name ABNF.
- o Clarified newline escaping rule.
- o Replaced one remaining "type" with "property".
- o Removed case insensitivity of parameter values.
- o XML property can now only contain one element that is not in the vCard 4 namespace.

- o Clarified interrelationship between LANGUAGE, cardinality, and PID.
- o Added "textphone" TEL type.
- o Fixed quoting of comma in ORG examples.

#### B.13. Changes in -10

- o Added component in SORT-STRING for sorting by given name. Fixed and expanded the examples.
- o Fixed KIND-value ABNF.
- o Fixed deprecated media type.
- o Created the CALSCALE parameter.
- o Strengthened the stance on character set.
- o Defined the Language-Tag ABNF.
- o Made explicit the fact that IANA does not register templates.
- o Created the XML property.
- o Added social networking examples to URL property.

#### B.14. Changes in -09

- o Removed special meaning for groups. Removed the "work" and "home" groups. Removed the group registry. Re-introduced the "work" and "home" TYPE parameter values. Applied the TYPE parameter to properties which supported the "work" and "home" groups.
- o Vendor namespace now uses private enterprise number in prefix.
- o Added "thing" value for KIND property.

#### B.15. Changes in -08

- o Allow 1985 (year only) in date ABNF.
- o Fixed missing country in ADR example.
- o Added the DATE-AND-OR-TIME value.

- o Made BDAY and DDAY use DATE-AND-OR-TIME.
- o Prefixed "param" and "value" production rules specific to properties with the property name.
- o Replaced the GENDER property with the SEX property.
- o Added the ANNIVERSARY property.
- o Added the "friend" and "spouse" types of RELATED.
- o TZ property now has text / uri value.
- o Refined the definitions of TITLE and ROLE.

B.16. Changes in -07

- o PREF is now bounded. 100 is the maximum value.
- o Added the "emergency" RELATED type.
- o Made GEO a URI.
- o Added GEO and TZ parameters to ADR.
- o Changed wording of "default" use of SOUND property.
- o Completely reworked the date, time, and date-time grammars.
- o Added the timestamp value type.
- o REV now has the timestamp value type.
- o Rewrote ABNF.
- o ORG can now have a single level.

B.17. Changes in -06

- o Corrected omission of resetability to text value for RELATED.
- o Let KEY value type be reset to a URI value.
- o ABNF fixes.
- o Made gender values extensible.

- o Gave the PREF parameter a positive integer value.
- o Removed usage of the undefined "word" ABNF production rule.
- o Defined property cardinalities.
- o Defined properties allowable in WORK and HOME groups.
- o Simplified the LANG property to use the vCard preference mechanism.
- o Created the "language-tag" value type.
- o Added PID to ABNF of SOURCE allowed parameters.
- o Clarified escaping rules.
- o Changed ABNF definition of non-standard X- properties.
- o Removed TYPE parameter from EMAIL properties in examples.
- o Created the CLIENTPIDMAP property.
- o Changed PID value to a pair of small integers.
- o Completely reworked synchronization mechanisms.
- o Created brand new synchronization example.

B.18. Changes in -05

- o Added multi PID value proposal.

B.19. Changes in -04

- o Added "location" value for KIND property.
- o Some fixes to ABNF.
- o Moved "pref" from being a TYPE value to a parameter in its own right.
- o Removed the "work" and "home" TYPE values.
- o Reintroduced the group construct.
- o Assigned meaning to WORK and HOME groups.

- o Restricted the TEL TYPE parameter value set.
- o In N property, removed additional names, and replaced with multiple given names.
- o Removed TYPE parameter from EMAIL and IMPP properties.
- o Replaced AGENT with a type of RELATED.
- o Use example.org domain in URL example.
- o Created initial IANA table of values.
- o Defined meaning of PUBLIC, PRIVATE, CONFIDENTIAL.

#### B.20. Changes in -03

- o Various changes to the synchronization mechanisms.
- o Allowed truncated format for dated. See issue #236.

#### B.21. Changes in -02

- o Removed useless text in IMPP description.
- o Added CalDAV-SCHED example to CALADRURI.
- o Removed CAPURI property.
- o Dashes in dates and colons in times are now mandatory.
- o Allow for dates such as 2008 and 2008-05 and times such as 07 and 07:54.
- o Removed inline vCard value.
- o Made AGENT only accept URI references instead of inline vCards.
- o Added the MEMBER property.
- o Renamed the UID parameter to PID.
- o Changed the value type of the PID parameter to "a small integer."
- o Changed the presence of UID and PID when synchronization is to be used from MUST to SHOULD.

- o Added the RELATED (Section 6.6.6) property.
- o Fixed many ABNF typos (issue #252).
- o Changed formatting of ABNF comments to make them easier to read (issue #226).

#### B.22. Changes in -01

- o Merged [RFC2739] in.
- o Converted all foobar.com, abc.com, etc. to example.com.
- o Fixed bugs in ABNF.
- o Made explicit that coordinates in the GEO property are expressed in the WGS 84 reference system.
- o Clarified folding issues with multi-byte characters.
- o Made the value of TEL a URI.
- o Added the UID parameter.
- o Made the UID property's value type a URI.
- o Added Section 7.
- o Created IANA process for registering new parameters, value types, and properties.
- o Created the initial IANA registries.
- o Created vendor namespace based on text from RFC 4288.

#### B.23. Changes in -00

- o Name change because draft has been accepted as WG item. Otherwise, same as draft-resnick-vcarddav-vcardrev-01.
- o Removed reference to RFC 2234.
- o Fixed errata from [http://www.rfc-editor.org/errata\\_search.php?rfc=2426](http://www.rfc-editor.org/errata_search.php?rfc=2426).
- o Removed passage referring to RFC 2425 profiles.



- o Renamed Section 6.4 from "Telecommunications Addressing Properties" to "Communications Properties."
- o Added Appendix A and Appendix B.
- o Added reference to [RFC4770].
- o Removed the group construct.
- o Made the N property no longer mandatory.
- o Added the KIND property.
- o Clarified meaning of TYPE parameter value for PHOTO, LOGO, KEY, and SOUND.
- o Removed the CONTEXT parameter.
- o Removed the MAILER property.
- o Made reference to [ISO9070] informative.
- o Removed "intl", "dom", "postal", and "parcel" TYPE parameter values for the ADR and LABEL properties.
- o Clarified meaning of "extended address" ADR field.
- o Mentioned [RFC3406] as another method of generating PROUID values.
- o Updated obsolete references.
- o Allowed BDAY and DDAY value types to be text values for fuzzy dates.
- o Removed the CHARSET property. Now the encoding is always UTF-8, except when overridden by the Content-Type (which is considered a compatibility feature).

Author's Address

Simon Perreault  
Viagenie  
2875 Laurier, suite D2-630  
Quebec, QC G1V 2M2  
Canada

Phone: +1 418 656 9254  
EMail: [simon.perreault@viagenie.ca](mailto:simon.perreault@viagenie.ca)  
URI: <http://www.viagenie.ca>

