Network Working Group	J. Smarr	
Internet-Draft	Plaxo	
Intended status: Informational	December 03, 2008	
Expires: June 6, 2009		

TOC

Portable Contacts: A Common Format and Protocol for Accessing Contacts
draft-smarr-vcarddav-portable-contacts-00

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.
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/1id-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 June 6, 2009.

Abstract

This document specifies Portable Contacts, XML and JSON address book document formats and an interface for accessing address book and friends-list information over HTTP.

Table of Contents

---

1. Introduction

TOC

The Portable Contacts specification is designed to make it easier for developers to give their users a secure way to access the address books and friends lists they have built up all over the web. Specifically, it seeks to create a common access pattern and contact schema that any site can provide, well-specified authentication and access rules, standard libraries that can work with any site, and absolutely minimal complexity, with the lightest possible toolchain requirements for developers.

By far the easiest way to start understanding this spec is to jump to the example in the Appendix. The format and meaning of the response should be readily apparent, and the majority of this document is merely an attempt to formalize the details of what should be relatively clear from this example.

This API defines a language- and platform- neutral protocol for Consumers to request address book, profile, and friends-list information from Service Providers. As a protocol, it is intended to be easy to understand and implement, either as a Service Provider or Consumer, using any language or platform of choice. It is also intended to be implemented by both individuals and small services as well as large providers, in any case where a service contains data about who a user knows and wishes to make that information portable, under the user's control.

While there are currently standards for describing contact info (such as vCard), these standards do not specify how to discover, access, and manipulate this information, and they do not capture the full range of information typically found in modern address book and social

networking applications. Several large companies have also released their own non-standard APIs for accessing and interacting with contact information, increasing the burden on developers and Consumers who wish to support most or all Service Providers. Nor do these APIs inform other providers as to how they should construct similar APIs. Thus Portable Contacts is an attempt to specify a complete, modern, and straight-forward recipe for Service Providers and Consumers of all sizes to make available and consume contact data in a standardized way.

1.1. Goals

TOC

The goal of Portable Contacts is to make it easier for developers to give their users a secure way to access the address books and friends lists they have built up all over the web. Specifically, we seek to create:

- A common access pattern and contact schema that any Service Provider can implement
- Well-specified authorization and access rules
- Free and open source libraries in many languages for most popular platforms
- Community-sourced support, documentation, and collaborative tools
- and absolutely minimal complexity, with the lightest possible toolchain requirements for developers.

A measure of our success will be the elimination of the "password anti-pattern," by making it far easier to implement Portable Contacts than to engage in scraping, as well as a dramatic increase in the number of sites that both provide and consume who-you-know data.

1.2. Approach

TOC

Our design is focused around ease of adoption, which means a few things:

1. First, our emphasis is on simplicity of design and targeted use cases, keeping our scope intentionally narrow at the outset. For example, version 1 is simply about access, and defers for now on the more complex issues around update and sync.
2. Second, we're taking a modern approach to who-you-know data by unifying traditional contact information and social network data, in order to properly represent the current diversity of the social web ecosystem.
3. Third, we're reusing existing standards wherever possible, including vCard, OpenSocial, XRDS-Simple, OAuth, etc.

4. And lastly, we're designing something that should be easy for current service providers to adopt. We started with a review of all the major existing contacts APIs and targeted common capabilities that they all share and provide. We believe this pragmatic balance is the best and quickest way to achieve our intended goal of widespread adoption.

1.3. Feedback

TOC

The Portable Contact specification is currently being developed on the http://groups.google.com/group/portablecontacts mailing list, with additional feedback coming from the OpenSocial community. Feedback can be posted to the Portable Contacts list or directly to the author. If you encounter any problems with joining the list, please contact the author.

---

2. Notational Conventions

TOC

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] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.).

---

3. Definitions

TOC

Contact: A record describing information about a particular person or entity, consisting of contact information (e.g. name, e-mail addresses, phone numbers) and other descriptive information, as is typically found in address book and social networking applications.

Service Provider: A web application that provides Contact information via the Portable Contacts protocol.

Consumer: A website or application that uses the Portable Contacts protocol to request contacts managed by the Service Provider.

Base URL: The root endpoint URL specified by the Service Provider during Discovery and used to make requests. Consumers MAY append

additional path information and query string parameters to this URL as part of the request.

Singular Field: A contact field that can appear at most once per contact, e.g. displayName or gender.

Plural Field: A contact field that can appear multiple times per contact, e.g. emails or tags.

Simple Field A Singular Field or Plural Field whose value is a single string attribute (see Section 7.1 (Structure)).

Complex Field A Singular Field or Plural Field whose value is an object containing multiple sub-field attributes (see Section 7.1 (Structure)).

Canonical Value: Specified string values for string-valued contact fields that represent common values in a canonical form, e.g. "male" and "female" for gender. Service Providers SHOULD conform to Canonical Values if appropriate, but MAY deviate if they need to represent additional values.

Primary Sub-Value: The sub-field in a Complex Field that should be used when sorting or filtering by that field. Unless otherwise specified, the value sub-field is always the Primary Sub-Field.

---

4. Workflow Overview

TOC

A Consumer wishing to access a user's data via Portable Contacts must start with an Initial Identifier for the Service Provider containing the user's data, usually provided by the user. In many cases, this may be the domain name of the Service Provider's web site, such as sample.site.org, but may be a more specific URL, such as the OpenID identifier of the user, if available. Consumers then perform Discovery on the Initial Identifier to determine where the Portable Contacts endpoint for this Service Provider resides. If successful, the Consumer may then attempt to request information from that endpoint. If the endpoint contains private data, the Service Provider will return an authorization challenge, and the Consumer must then guide the user through an appropriate authorization flow to obtain the credentials necessary to access this private data. Upon successful authorization, the Consumer may request data from the Portable Contacts endpoint using these authorization credentials. Whether accessing public or private data, Consumers may request a specific subset of the user's data using standard Query Parameters. Upon a successful request, the data is returned in the response, and the Consumer may then parse the response data and use it as desired. The following sections detail each of these steps.

TOC

5. Discovery

Portable Contacts API endpoint is discoverable from the domain root using [XRDS-Simple] (Hammer-Lahav, E., “XRDS-Simple 1.0,” .) (previously known as YADIS). The API is identified by the Service Type http://portablecontacts.net/spec/1.0 and the corresponding URI is the Base URL for the API. The Base URL MUST NOT contain any query string, as additional path information and query string variables MAY be appended by Consumers as part of forming the request (as described in detail below).
An example XRDS-Simple document describing the availability and location of a Portable Contacts endpoint might look like this:

```
<XRDS xmlns="xri://$xrds">
  <XRD xmlns:simple="http://xrds-simple.net/core/1.0"
      xmlns="xri://$XRD*($v*2.0)" version="2.0">
    <Type>xri://$xrds*simple</Type>
    <Service>
      <Type>http://portablecontacts.net/spec/1.0</Type>
      <URI>http://sample.site.org/path/to/api/</URI>
    </Service>
  </XRD>
</XRDS>
```

In addition to discovering the endpoint itself, Service Providers using OAuth to protect responses MUST also support OAuth Discovery, as described in Section 6.1 (Authentication and Authorization).

---

6. Invocation

TOC

All requests to the Service Provider are made as HTTP GET operations on a URL deriving from the Base URL specified in Section 5 (Discovery). Consumers MAY append additional path information and/or query string parameters to the Base URL as part of the request, as specified in Section 6.3 (Query Parameters). Additionally, authentication information MAY be sent via POST data or additional HTTP headers in the request, as specified in Section 6.1 (Authentication and Authorization). Responses are returned in the body of the HTTP response, formatted as JSON or XML, depending on what is requested. Response and error codes SHOULD be transmitted via the HTTP status code of the response (if possible), and SHOULD also be specified in the body of the response, as described in Section 6.4 (Response Format) and Section 6.5 (Error Codes). Since the API endpoint is dynamic (and not serving static content), Consumers MUST NOT interpret any cache headers in the response as having meaning concerning when the same URL request might return a different response upon subsequent invocation.

6.1. Authentication and Authorization

TOC

The data returned by a Portable Contacts endpoint MAY contain public data, or it MAY contain private data. If the data returned is public, no authentication or authorization is required. In most cases however,

the data returned is not public, and Service Providers SHOULD ensure that the user has given prior consent, either explicitly or implicitly, for their information to be released by this API. Typically this is done by Consumers obtaining either Direct Authorization (with raw credentials, for example the user's username and password) or Delegated Authorization (with an access token obtained out-of-band by the user, and given to the Consumer to present as part of the request). Portable Contacts specifies standard mechanisms for both types of authorization, so that Consumers may be able to obtain private data on a user's behalf from Service Providers in an automated and consistent fashion. Regardless of the Authorization method used, the context of the request (i.e. the user for whom data is being requested) MUST be inferred by Service Providers from the Base URL and the authorization credentials provided. If public data is being accessed (and no authorization is provided), the Base URL MUST contain enough information for Service Providers to know which data to return, but if private data is being accessed (and authorization is provided), the same Base URL MAY return information for different users depending on the authorization credentials provided.

6.1.1. Delegated Authorization

TOC

Service Providers wishing to provide Delegated Authorization MUST support [OAuth Core 1.0] (OAuth, OCW., “OAuth Core 1.0,” .) as an OAuth Service Provider, and MAY also support additional Delegated Authorization mechanisms, if they choose. Service Providers supporting OAuth MUST also support [OAuth Discovery] (Eran Hammer-Lahav, E., “OAuth Discovery 1.0,” .) to facilitate automatic discovery of authorization endpoints for Consumers. Service Providers SHOULD provide a mechanism for Consumers to automatically obtain a Consumer Key and Consumer Secret, but MAY require this to be done out-of-band.

6.1.2. Direct Authorization

TOC

Service Providers wishing to provide Direct Authorization MUST support HTTP Basic Access Authentication [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.), and MAY also support additional Direct Authorization mechanisms, if they choose. In addition to being a well-established mechanism for Direct Authorization, HTTP Basic has the added benefit of being understood by most Web Browsers, and can prompt users to enter their credentials as part of accessing a resource protected in this manner. There are also convenient ways of providing and parsing HTTP Basic credentials in popular tools and libraries like curl and PHP.

TOC

6.1.3. Available Authorization Methods

Service Providers that provide access to private data MAY choose not to support either Direct Authorization or Delegated Authorization, depending on their security requirements, but they MUST support either OAuth or HTTP Basic auth if they require any Authorization. When accessing a Portable Contacts endpoint, if sufficient authorization credentials are not provided, the Service Provider SHOULD return a 401 Unauthorized response, and SHOULD provide the available Authorization mechanisms available by including WWW-Authenticate headers in the response for each type of Authorization method supported (as defined in [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," June 1999.), section 14.47. Consumers will then be able to recognize that the API is a protected resource and initiate the proper Authorization process needed to obtain the appropriate credentials. An example set of WWW-Authenticate headers returned by a Service Provider that supports both OAuth and HTTP Basic might look like this. Note that the realm value is intended to be an opaque string that merely defines a shared label for resources that share the same authorization requirements.

```
WWW-Authenticate: OAuth realm="sample.site.org"
WWW-Authenticate: Basic realm="sample.site.org"
```

If Service Providers wish to make some response data publicly available and also provide additional info given the proper authorization credentials, they SHOULD provide a 200 OK response to requests without authorization with a WWW-Authenticate header in the response indicating that additional info is available via the specified authorization mechanisms.

6.2. Additional Path Information

TOC

A request using the Base URL alone MUST yield a result, assuming that adequate authorization credentials are provided. In addition, Consumers MAY append additional path information to the Base URL to request more specific information. Service Providers MUST recognize the following additional path information when appended to the Base URL, and MUST return the corresponding data:

* /@me/@all -- Return all contact info (equivalent to providing no additional path info)

* /@me/@all/{id} -- Only return contact info for the contact whose id value is equal to the provided {id}, if such a contact exists. In this case, the response format is the same as when requesting all contacts, but any contacts not matching the requested ID MUST be filtered out of the result list by the Service Provider

* /@me/@self -- Return contact info for the owner of this information, i.e. the user on whose behalf this request is being made. In this case, the response format is the same as when requesting all contacts, but any contacts not matching the

requested ID MUST be filtered out of the result list by the Service Provider.

6.3. Query Parameters

TOC

Portable Contacts defines a standard set of operations that can be used to filter, sort, and paginate response results. The operations are specified by adding query parameter to the Base URL, either in the query string or as HTTP POST data. Providers MAY support additional query parameters not specified here, and Providers SHOULD ignore any query parameters they don't recognize.

6.3.1. Filtering

TOC

Filtering is used to limit the request results to Contacts that match given criteria. Content filtering is accomplished by combining three request parameters:

filterBy: Specifies the field name to filter by. If the specified field is a Plural Field, the Contact SHALL match if any of the instances of the given field match the specified criterion (e.g. if a contact has multiple `emails` values, only one has to match for the entire Contact to match). If a Simple Field is specified, its value must match the specified `filterValue` according to the specified `filterOp`. If a Complex Field is specified, its Primary Sub-Field must match. If the specified field is not a direct child of the `entry` element, the full path MUST be specified using the '.' character as separator. For example, to filter by gender the parameter value is `gender` and to filter by first name, the parameter value is `name.givenName`.

filterOp: Specifies the comparison method used to evaluate the field value with the value of the filter criterion. Providers SHOULD support the following values:

* `equals`: the two values must be identical strings.
* `contains`: the entire `filterValue` must be a substring of the Contact field value.
* `startswith`: the entire `filterValue` must be a substring of the Contact field value, starting at the beginning of the field value. This criterion is satisfied if the two strings are equal.
* `present`: a Contact matches the criterion if the field specified by `filterBy` has a non-empty value, or if it contains a non-empty node for complex fields.

> Providers MAY support additional filter operations if they choose. Providers MUST decline to filter results if the specified filter operation is not recognized (as per Section 6.3.5 (Declining to honor query parameters)).
>
> **filterValue:** Specifies the value to filter by, using the comparison method defined by filterOp.

In addition, requests can filter content based on their update timestamp:

> **updatedSince:** Returns only contacts that have been modified on or after the given time, specified as an xs:dateTime. The filter is based on the value of the updated field, and can be used independently of other filters or combined. It enables a basic syndication pattern when accessing the same data over time. The first API call returns all data, which can be stored locally. Subsequent API calls can specify updatedSince with the time of the last API call, so that only contacts that have been added or modified since the last API call will be returned.

Here are a few illustrative examples of filtering matches with filterBy, filterOp, and filterValue. In each case, assume the following two contacts would be returned if no filtering parameters were provided:

```
{
  "id": "1",
  "displayName": "Chris Messina",
  "urls": [
    { "value": "http://factoryjoe.com/blog", "type": "blog" }
  ]
},
{
  "id": "2",
  "displayName": "Joseph Smarr",
  "emails": [
    { "value": "joseph@plaxo.com", "type": "work", "primary": "true" },
    { "value": "jsmarr@gmail.com", "type": "home" }
  ],
}
```

Given the parameters filterBy=displayName&filterOp=startswith&filterValue=Chr, only the first contact (with id=1) would match and be returned. However, with parameters filterBy=displayName&filterOp=present, both contacts would be returned. Given the parameters filterBy=email&filterOp=contains&filterValue=plaxo.com, only the second contact (with id=2) would match, as would it be the only contact to match given the parameters filterBy=email&filterOp=present.
If a request specifies a filterValue but no filterBy or filterOp, it is up to the Provider how to interpret this filter request. Providers MAY choose to default to filtering by a given field (e.g. displayName); they MAY choose to implement a custom, Provider-specific query syntax for filterValue in this case; or they MAY choose to reject requests of this type. In general, if Consumers want to request specific behavior

from Providers, they should do so by being explicit in their use of query parameters.

6.3.2. Sorting

TOC

Sorting allows requests to specify the order in which contacts are returned.

sortBy Specifies the field name whose value SHALL be used to order the returned Contacts. The sort order is determine by the sortOrder parameter. If sortBy is a Singular Field, contacts are sorted according to that field's value; if it's a Plural Field, contacts are sorted by the Value (or Major Value, if it's a Complex Field, see Section 7 (Contact Schema)) of the field marked with "primary": "true", if any, or else the first value in the list, if any, or else they are sorted last if the given contact has no data for the given field.

sortOrder The order in which the sortBy parameter is applied. Allowed values are ascending and descending. If a value for sortBy is provided and no sortOrder is specifies, the sortOrder SHALL default to ascending. Sort order is expected to be case-insensitive Unicode alphabetic sort order, with no specific locale implied.

6.3.3. Pagination

TOC

The pagination parameters can be used together to "page through" a large number of results in manageable chunks:

startIndex: Specifies the offset of the first result to be returned with respect to the list of contacts that would be returned if no startIndex were provided. For instance, if in a given request 10 contacts would normally be provided, if startIndex is 7 and no count is specified, then only the last 3 contacts in that list would be returned (contacts are zero-indexed). If startIndex is greater than or equal to the total number of results that would be returned, no contacts are returned. Value MUST be a non-negative integer and defaults to 0 if no value is specified.

count: If non-zero, specifies the maximum number of contacts the Consumer would like the Provider to return at a time. Value MUST be a non-negative integer and defaults to 0 if no value is specified. A count of 0 means that is up to the Provider to determine how many contacts to return by default (some Providers may return all contacts by default; others may return a fixed default number like 10). Providers SHOULD honor a very large count value, and SHOULD support returning all contacts at once when presented with a count request that is larger than the number of contacts the user has, but Providers MAY choose to

never return more than a Provider-determined maximum number of contacts per request, if returning all contacts is too burdensome. In all cases, at most count contacts SHALL be returned, starting at startIndex and counting up from there. In each of these cases, Providers MUST indicate the total number of contacts they chose to return in the response using the itemsPerPage response element (see Section 6.4 (Response Format)).

For instance, on an initial query, specifying startIndex=0&count=10 will return only the first 10 results. The total number of possible results is indicated by the totalResults field of results, so the client knows how many "pages" of results exist. A subsequent query of startIndex=10&count=10 will return the next 10 results, and so on.

6.3.4. Presentation

TOC

Presentation controls the format, makeup, and delivery mechanism for returning the requested result set:

fields: If non-empty, each contact returned SHALL contain only the fields explicitly requested. Service Provider MAY return a subset of the requested fields if they are not supported. This field is used for efficiency when the client only wishes to access a subset of the fields normally returned in results. Value is a comma separated list of top level field names (e.g. id,name,emails,photos) and defaults to an empty list which means it's up to the Provider which fields to return. Consumers may request all available fields to be returned by using the special value @all.

format: Specifies the format in which the response data is returned. Service Providers MUST support the values json for JSON (http://json.org) and xml for XML (http://www.w3.org/XML/) and MAY support additional formats if desired. The format defaults to json if no format is specified. The data structure returned is equivalent in both formats; the only difference is in the encoding of the data. Singular Fields are encoded as string key/value pairs in JSON and tags with text content in XML, e.g. "field": "value" and <field>value</field> respectively. Plural Fields and Plural Bundles are encoded as arrays in JSON and repeated tags in XML, e.g. "fields": ["value1", "value2"] and <fields>value1</fields><fields>value2</fields> respectively. Nodes with multiple sub-nodes are represented as objects in JSON and tags with sub-tags in XML, e.g. "field": { "subfield1": "value1", "subfield2": "value2" } and <field><subfield1>value1</subfield1><subfield2>value2</subfield2></field> respectively.

TOC

6.3.5. Declining to honor query parameters

Providers SHOULD honor all filtering, sorting, and pagination requests specified via Query Parameters. However, in some instances it may be too burdensome to comply with a particular request, e.g. because the Provider does not have an efficient database index set up for a given field that is requested for filtering or sorting, and is unable to efficiently fetch all data and post-process the results to honor the request before returning the response. In such cases, Providers MAY decline to honor the request (or specific pieces of the request). If any part of the request is declined, Providers MUST specify which part(s) of the request were declined in the response, using "sorted": false, "filtered": false, and/or "updatedSince": false as appropriate. For efficiency, Providers SHOULD omit these response fields if that part of the request was successfully performed, or if no such Query Parameter was specified in the request.
Note that since all of the filtering, sorting, and pagination operations are designed to reduce the amount of data returned, it is possible for Consumers to emulate these operations client-side when a Provider declines to perform them server-side. For instance, filtering can be accomplished by iterating through each entry returned and deleting those that do not match the filtering criteria. Thus Consumers can request these operations to be performed server-side, and Providers will honor them if possible, and otherwise indicate to Consumers that they need to be performed client-side, effectively "splitting the workload" while maintaining consistent semantics.

6.4. Response Format

TOC

The structure of the response object returned from a successful request is as follows. The root element is response, which is shown explicitly as the root element in XML format, and is the anonymous root object returned when the format is JSON (i.e. in JSON, the response returned is the object value of the response node). The response node MUST contain the following child nodes, and MAY contain additional nodes that the Service Provider wishes to add to expose additional data. Note that startIndex, itemsPerPage, and totalResults are based on [OpenSearch] (Clinton, D., “OpenSearch 1.1,” .). See the Appendix for a full example.

* startIndex: the index of the first result returned in this response, relative to the starting index of all results that would be returned if no startIndex had been requested. In general, this will be equal to the value requested by the startIndex, or 0 if no specific startIndex was requested.

* itemsPerPage: the number of results returned per page in this response. In general, this will be equal to the count Query Parameter, but MAY be less if the Service Provider is unwilling to return as many results per page as requested, or if there are less than the requested number of results left to return when starting at the current startIndex. This field MUST be present if and only if a value for count is specified in the request.

* totalResults: the total number of contacts that would be returned if there were no startIndex or count specified. This value tells the Consumer how many total results to expect, regardless of the current pagination being used, but taking into account the current filtering options in the request.

* entry: an array of Contact objects, one for each contact matching the request, as defined in Section 7.2 (entry Element). For consistency of parsing, if the request could possibly return multiple contacts (as is normally the case), this value MUST always be an array of results, even if there happens to be 0 or 1 matching results. If the request is specifically for a single contact (e.g. because the request contains Additional Path Information like /@me/@all/{id} or /@me/@self), then entry MUST be an object containing the single contact returned (i.e. "entry": [{ /* first contact */ }] and "entry": { /* only contact */ } respectively).

6.5. Error Codes

TOC

The Service Provider MUST return a response code with every response. Response codes are numeric and conform to existing HTTP response codes where possible, as defined below. In addition to the response code, Service Providers SHOULD also provide a human-readable reason that explains the reason for the response code. This message SHOULD be intelligible to developers, but MAY be unsuitable for display to end-users. Clients SHOULD provide their own appropriate error message to users when encountering an error response.
Service Providers SHOULD conform to the following response codes to indicate the following situations. Service Providers MAY return additional codes to indicate additional information, but are discouraged from doing so and should instead augment the reason text with existing codes, if possible.

200: OK (response returned successfully)

400: Bad Request (request was malformed or illegal and cannot be completed)

401: Unauthorized (authentication headers / parameters were invalid or missing)

404: Not Found (the request points to an object that does not exist, e.g. to an unknown contact id; note that Service Providers MUST return a 200 with an empty array of contacts if the request has filtering parameters that are valid but have no matches)

500: Internal Server Error (un unexpected error occurred during processing)

503: Service Unavailable (service is temporarily unavailable; this may be because the Consumer has exceeded their rate-limit of requests)

7. Contact Schema

TOC

The Contact schema defines the containers and attributes used to deliver an individual Contact or a list of Contacts as requested by the Consumer. The traditional contact info fields were taken directly from the vCard spec where possible [RFC2425] (Howes, T., Smith, M., and F. Dawson, "A MIME Content-Type for Directory Information," September 1998.), though instances of vCard's archaic spellings were modernized (e.g. addresses instead of adr). Even with the spelling updates, the field mappings remain equivalent, which means it should be easy to convert Portable Contacts data to and from vCard. By convention, Singular Fields have singular spelling (e.g. displayName) and plural fields have plural spelling (e.g. phoneNumbers) to make it easy to distinguish them.

Each contact returned MUST include the id and displayName fields with non-empty values, but all other fields are optional, and it is recognized that not all Service Providers will be able to provide data for all the supported fields. The field list below is broad so that, for Service Providers that do support any of these fields, there is a standard field name available.

7.1. Structure

TOC

Each field is defined as either a Singular Field, in which case there MUST NOT be more than one instance of that field per contact, or as a Plural Field, in which case any number of instances of that field MAY be present per contact.

Contact information is formatted using labeled attributes with either structured or unstructured string data. Each attribute value consists of one of the following types:

> **Simple:** A single string attribute which MAY specify a REQUIRED data format or allow any string. A simple field MAY contain Canonical Values specified, in which case Service Providers SHOULD try to conform to those values if appropriate, but MAY provide alternate string values to represent additional values.
>
> **Boolean:** A special case of a Simple Field with two legal values: true and false. Values are case-sensitive.
>
> **Complex:** A multi-value attribute containing any combination of other attributes. Complex attributes are defined by listing the child attributes and their types. For most Complex Fields, the value sub-field contains the Major Value of that field (i.e. the primary piece of contact information described by that field), and the other fields provide additional meta-data.

7.2. entry Element

TOC

Unless otherwise specified, all fields are optional and of type xs:string. Also, unless specified, all field values MUST NOT contain any newline characters (\r or \n).

7.2.1. Singular Fields

TOC

id: Unique identifier for the Contact. Each Contact returned MUST include a non-empty id value. This identifier MUST be unique across this user's entire set of Contacts, but MAY not be unique across multiple users' data. It MUST be a stable ID that does not change when the same contact is returned in subsequent requests. For instance, an e-mail address is not a good id, because the same person may use a different e-mail address in the future. Usually, in internal database ID will be the right choice here, e.g. "12345".

displayName: The name of this Contact, suitable for display to end-users. Each Contact returned MUST include a non-empty displayName value. The name SHOULD be the full name of the Contact being described if known (e.g. Joseph Smarr or Mr. Joseph Robert Smarr, Esq.), but MAY be a username or handle, if that is all that is available (e.g. jsmarr). The value provided SHOULD be the primary textual label by which this Contact is normally displayed by the Service Provider when presenting it to end-users.

name: The broken-out components and fully formatted version of the contact's real name, as described in Section 7.3 (name Element).

nickname: The casual way to address this Contact in real life, e.g. "Bob" or "Bobby" instead of "Robert". This field SHOULD NOT be used to represent a user's username (e.g. jsmarr or daveman692); the latter should be represented by the preferredUsername field.

published: The date this Contact was first added to the user's address book or friends list (i.e. the creation date of this entry). The value MUST be a valid xs:dateTime (e.g. 2008-01-23T04:56:22Z).

updated: The most recent date the details of this Contact were updated (i.e. the modified date of this entry). The value MUST be a valid xd:dateTime (e.g. 2008-01-23T04:56:22Z). If this Contact has never been modified since its initial creation, the value MUST be the same as the value of published. Note the updatedSince Query Parameter described in Section 6.3 (Query Parameters) can be used to select only contacts whose updated value is equal to or more recent than a given xs:dateTime. This enables Consumers

> to repeatedly access a user's data and only request newly added or updated contacts since the last access time.
>
> **birthday:** The birthday of this contact. The value MUST be a valid xs:date (e.g. 1975-02-14. The year value MAY be set to 0000 when the age of the Contact is private or the year is not available.
>
> **anniversary:** The wedding anniversary of this contact. The value MUST be a valid xs:date (e.g. 1975-02-14. The year value MAY be set to 0000 when the year is not available.
>
> **gender:** The gender of this contact. Service Providers SHOULD return one of the following Canonical Values, if appropriate: male, female, or undisclosed, and MAY return a different value if it is not covered by one of these Canonical Values.
>
> **note:** Notes about this contact, with an unspecified meaning or usage (normally contact notes by the user about this contact). This field MAY contain newlines.
>
> **preferredUsername:** The preferred username of this contact on sites that ask for a username (e.g. jsmarr or daveman692). This field may be more useful for describing the owner (i.e. the value when /@me/@self is requested) than the user's contacts, e.g. Consumers MAY wish to use this value to pre-populate a username for this user when signing up for a new service.
>
> **utcOffset:** The offset from UTC of this Contact's current time zone, as of the time this response was returned. The value MUST conform to the offset portion of xs:dateTime, e.g. -08:00. Note that this value MAY change over time due to daylight saving time, and is thus meant to signify only the current value of the user's timezone offset.
>
> **connected:** Boolean value indicating whether the user and this Contact have established a bi-directionally asserted connection of some kind on the Service Provider's service. The value MUST be either true or false. The value MUST be true if and only if there is at least one value for the relationship field, described below, and is thus intended as a summary value indicating that some type of bi-directional relationship exists, for Consumers that aren't interested in the specific nature of that relationship. For traditional address books, in which a user stores information about other contacts without their explicit acknowledgment, or for services in which users choose to "follow" other users without requiring mutual consent, this value will always be false.

The following additional Singular Fields are defined, based on their specification in OpenSocial [OpenSocial] (Panzer, J., “OpenSocial 0.8.1 RESTful Protocol Specification,” .): aboutMe, bodyType, currentLocation, drinker, ethnicity, fashion, happiestWhen, humor, livingArrangement, lookingFor, profileSong, profileVideo, relationshipStatus, religion, romance, scaredOf, sexualOrientation, smoker, and status.

7.2.2. Plural Fields

TOC

Unless specified otherwise, all Plural Fields have the same three standard sub-fields:

value: The primary value of this field, e.g. the actual e-mail address, phone number, or URL. When specifying a sortBy field in the Query Parameters for a Plural Field, the default meaning is to sort based on this value sub-field. Each non-empty Plural Field value MUST contain at least the value sub-field, but all other sub-fields are optional.

type: The type of field for this instance, usually used to label the preferred function of the given contact information. Unless otherwise specified, this string value specifies Canonical Values of work, home, and other.

primary: A Boolean value indicating whether this instance of the Plural Field is the primary or preferred value of for this field, e.g. the preferred mailing address or primary e-mail address. Service Providers MUST NOT mark more than one instance of the same Plural Field as primary="true", and MAY choose not to mark any fields as primary, if this information is not available. For efficiency, Service Providers SHOULD NOT mark all non-primary fields with primary="false", but should instead omit this sub-field for all non-primary instances.

When returning Plural Fields, Service Providers SHOULD canonicalize the value returned, if appropriate (e.g. for e-mail addresses and URLs). Providers MAY return the same value more than once with different types (e.g. the same e-mail address may used for work and home), but SHOULD NOT return the same (type, value) combination more than once per Plural Field, as this complicates processing by the Consumer.

emails: E-mail address for this Contact. The value SHOULD be canonicalized by the Service Provider, e.g. joseph@plaxo.com instead of joseph@PLAXO.COM.

urls: URL of a web page relating to this Contact. The value SHOULD be canonicalized by the Service Provider, e.g. http://josephsmarr.com/about/ instead of JOSEPHSMARR.COM/about/. In addition to the standard Canonical Values for type, this field also defines the additional Canonical Values blog and profile.

phoneNumbers: Phone number for this Contact. No canonical value is assumed here. In addition to the standard Canonical Values for type, this field also defines the additional Canonical Values mobile, fax, and pager.

ims: Instant messaging address for this Contact. No official canonicalization rules exist for all instant messaging addresses, but Service Providers SHOULD remove all whitespace and convert the address to lowercase, if this is appropriate for the service this IM address is used for. Instead of the standard Canonical Values for type, this field defines the following Canonical

Values to represent currently popular IM services: aim, gtalk, icq, xmpp, msn, skype, qq, and yahoo.

photos: URL of a photo of this contact. The value SHOULD be a canonicalized URL, and MUST point to an actual image file (e.g. a GIF, JPEG, or PNG image file) rather than to a web page containing an image. Service Providers MAY return the same image at different sizes, though it is recognized that no standard for describing images of various sizes currently exists. Note that this field SHOULD NOT be used to send down arbitrary photos taken by this user, but specifically profile photos of the contact suitable for display when describing the contact.

tags: A user-defined category or label for this contact, e.g. "favorite" or "web20". These values SHOULD be case-insensitive, and there SHOULD NOT be multiple tags provided for a given contact that differ only in case. Note that this field is a Simple Field, meaning each instance consists only of a string value.

relationships: A bi-directionally asserted relationship type that was established between the user and this contact by the Service Provider. The value SHOULD conform to one of the XFN relationship values (e.g. kin, friend, contact, etc.) if appropriate, but MAY be an alternative value if needed. Note this field is a parallel set of category labels to the tags field, but relationships MUST have been bi-directionally confirmed, whereas tags are asserted by the user without acknowledgment by this Contact. Note that this field is a Simple Field, meaning each instance consists only of a string value.

addresses: A physical mailing address for this Contact, as described in Section 7.4 (address Element).

organizations: A current or past organizational affiliation of this Contact, as described in Section 7.5 (organization Element).

accounts: An online account held by this Contact, as described in Section 7.6 (account Element).

The following additional Plural Fields are defined, based on their specification in OpenSocial: activities, books, cars, children, food, heroes, interests, jobInterests, languages, languagesSpoken, movies, music, pets, politicalViews, quotes, sports, turnOffs, turnOns, and tvShows.

7.3. name Element

TOC

The components of the contact's real name. Providers MAY return just the full name as a single string in the formatted sub-field, or they MAY return just the individual component fields using the other sub-fields, or they MAY return both. If both variants are returned, they SHOULD be describing the same name, with the formatted name indicating how the component fields should be combined.

formatted:
The full name, including all middle names, titles, and suffixes as appropriate, formatted for display (e.g. Mr. Joseph Robert Smarr, Esq.). This is the Primary Sub-Field for this field, for the purposes of sorting and filtering.

familyName: The family name of this Contact, or "Last Name" in most Western languages (e.g. Smarr given the full name Mr. Joseph Robert Smarr, Esq.).

givenName: The given name of this Contact, or "First Name" in most Western languages (e.g. Joseph given the full name Mr. Joseph Robert Smarr, Esq.).

middleName: The middle name(s) of this Contact (e.g. Robert given the full name Mr. Joseph Robert Smarr, Esq.).

honorificPrefix: The honorific prefix(es) of this Contact, or "Title" in most Western languages (e.g. Mr. given the full name Mr. Joseph Robert Smarr, Esq.).

honorificSuffix: The honorifix suffix(es) of this Contact, or "Suffix" in most Western languages (e.g. Esq. given the full name Mr. Joseph Robert Smarr, Esq.).

7.4. address Element

TOC

The components of a physical mailing address. Service Providers MAY return just the full address as a single string in the formatted sub-field, or they MAY return just the individual component fields using the other sub-fields, or they MAY return both. If both variants are returned, they SHOULD be describing the same address, with the formatted address indicating how the component fields should be combined.

formatted: The full mailing address, formatted for display or use with a mailing label. This field MAY contain newlines. This is the Primary Sub-Field for this field, for the purposes of sorting and filtering.

streetAddress: The full street address component, which may include house number, street name, PO BOX, and multi-line extended street address information. This field MAY contain newlines.

locality: The city or locality component.

region: The state or region component.

postalCode: The zipcode or postal code component.

country: The country name component.

7.5. organization Element

TOC

Describes a current or past organizational affiliation of this contact. Service Providers that support only a single Company Name and Job Title field should represent them with a single `organization` element with `name` and `title` properties, respectively.

> **name:** The name of the organization (e.g. company, school, or other organization). This field MUST have a non-empty value for each organization returned. This is the Primary Sub-Field for this field, for the purposes of sorting and filtering.
>
> **department:** The department within this organization.
>
> **title:** The job title or role within this organization.
>
> **type:** The type of organization, with Canonical Values `job` and `school`.
>
> **startDate:** The date this Contact joined this organization. This value SHOULD be a valid xs:date if possible, but MAY be an unformatted string, since it is recognized that this field is often presented as free-text.
>
> **endDate:** The date this Contact left this organization or the role specified by title within this organization. This value SHOULD be a valid xs:date if possible, but MAY be an unformatted string, since it is recognized that this field is often presented as free-text.
>
> **location:** The physical location of this organization. This may be a complete address, or an abbreviated location like "San Francisco".
>
> **description:** A textual description of the role this Contact played in this organization. This field MAY contain newlines.

7.6. account Element

TOC

Describes an account held by this Contact, which MAY be on the Service Provider's service, or MAY be on a different service. Consumers SHOULD NOT assume that this account has been verified by the Service Provider to actually belong to this Contact. For each account, the `domain` is the top-most authoritative domain for this account, e.g. `yahoo.com` or `reader.google.com`, and MUST be non-empty. Each account must also contain a non-empty value for either `username` or `userid`, and MAY contain both, in which case the two values MUST be for the same account. These accounts can be used to determine if a user on one service is also known to be the same person on a different service, to facilitate connecting to people the user already knows on different services. If Consumers want to turn these accounts into profile URLs, they can use

an open-source library like [google-sgnodemapper] (Fitzpatrick, B., "Social Graph Node Mapper," .).

domain: The top-most authoritative domain for this account, e.g. "twitter.com". This is the Primary Sub-Field for this field, for the purposes of sorting and filtering.

username: An alphanumeric user name, usually chosen by the user, e.g. "jsmarr".

userid: A user ID number, usually chosen automatically, and usually numeric but sometimes alphanumeric, e.g. "12345" or "1Z425A".

---

8. IANA Considerations

TOC

This memo includes no request to IANA at this time. [[Future drafts are likely to request registration for the XML and JSON content types.]]

---

9. Security Considerations

TOC

This memo abides by the security considerations of HTTP Basic Auth [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," June 1999.) and the OAuth protocol [OAuth Core 1.0] (OAuth, OCW., "OAuth Core 1.0," .).

Appendix A. Example

TOC

Here is a sample request and response that illustrates much of Portable Contacts. For simplicity, authorization information is not shown in the request.
Sample request (via HTTP GET):

```
http://sample.site.org/path/to/api/@me/@all?startIndex=10
    &count=10&sortBy=displayName
```

Sample response (JSON):

```
{
  "startIndex": 10,
  "itemsPerPage": 10,
  "totalResults": 12,
  "entry": [
    {
      "id": "123",
      "displayName": "Minimal Contact"
    },
    {
      "id": "703887",
      "displayName": "Mork Hashimoto",
      "name": {
        "familyName": "Hashimoto",
        "givenName": "Mork"
      },
      "birthday": "0000-01-16",
      "gender": "male",
      "drinker": "heavily",
      "tags": [
        "plaxo guy"
      ],
      "emails": [
        {
          "value": "mhashimoto-04@plaxo.com",
          "type": "work",
          "primary": "true"
        },
        {
          "value": "mhashimoto-04@plaxo.com",
          "type": "home"
        },
        {
          "value": "mhashimoto@plaxo.com",
          "type": "home"
        }
      ],
      "urls": [
        {
          "value": "http://www.seeyellow.com",
          "type": "work"
        },
        {
          "value": "http://www.angryalien.com",
          "type": "home"
        }
      ],
      "phoneNumbers": [
        {
          "value": "KLONDIKE5",
          "type": "work"
        },
        {
          "value": "650-123-4567",
          "type": "mobile"
        }
      ],
```

```
          "photos": [
            {
              "value": "http://sample.site.org/photos/12345.jpg",
              "type": "thumbnail"
            }
          ],
          "ims": [
            {
              "value": "plaxodev8",
              "type": "aim"
            }
          ],
          "addresses": [
            {
              "type": "home",
              "streetAddress": "742 Evergreen Terrace\nSuite 123",
              "locality": "Springfield",
              "region": "VT",
              "postalCode": "12345",
              "country": "USA",
              "formatted":
              "742 Evergreen Terrace\nSuite 123\nSpringfield, VT 12345 USA"
            }
          ],
          "organizations": [
            {
              "name": "Burns Worldwide",
              "title": "Head Bee Guy"
            }
          ],
          "accounts": [
            {
              "domain": "plaxo.com",
              "userid": "2706"
            }
          ]
        }
      ]
    }
```


Sample response (XML):

```
<response>
 <startIndex>10</startIndex>
 <itemsPerPage>10</itemsPerPage>
 <totalResults>12</totalResults>
 <entry>
  <id>123</id>
  <displayName>Minimal Contact</displayName>
 </entry>
 <entry>
  <id>703887</id>
  <displayName>Mork Hashimoto</displayName>
  <name>
   <familyName>Hashimoto</familyName>
   <givenName>Mork</givenName>
  </name>
  <birthday>0000-01-16</birthday>
  <gender>male</gender>
  <drinker>heavily</drinker>
  <tags>plaxo guy</tags>
  <emails>
   <value>mhashimoto-04@plaxo.com</value>
   <type>work</type>
   <primary>true</primary>
  </emails>
  <emails>
   <value>mhashimoto-04@plaxo.com</value>
   <type>home</type>
  </emails>
  <emails>
   <value>mhashimoto@plaxo.com</value>
   <type>home</type>
  </emails>
  <urls>
   <value>http://www.seeyellow.com</value>
   <type>work</type>
  </urls>
  <urls>
   <value>http://www.angryalien.com</value>
   <type>home</type>
  </urls>
  <phoneNumbers>
   <value>KLONDIKE5</value>
   <type>work</type>
  </phoneNumbers>
  <phoneNumbers>
   <value>650-123-4567</value>
   <type>mobile</type>
  </phoneNumbers>
  <photos>
   <value>http://sample.site.org/photos/12345.jpg</value>
   <type>thumbnail</type>
  </photos>
  <ims>
   <value>plaxodev8</value>
   <type>aim</type>
  </ims>
  <addresses>
```

```
    <type>home</type>
    <streetAddress><![CDATA[742 Evergreen Terrace
Suite 123]]></streetAddress>
    <locality>Springfield</locality>
    <region>VT</region>
    <postalCode>12345</postalCode>
    <country>USA</country>
    <formatted><![CDATA[742 Evergreen Terrace
Suite 123
Springfield, VT 12345 USA]]></formatted>
   </addresses>
   <organizations>
    <name>Burns Worldwide</name>
    <title>Head Bee Guy</title>
   </organizations>
   <accounts>
    <domain>plaxo.com</domain>
    <userid>2706</userid>
   </accounts>
  </entry>
</response>
```

Appendix B. Compatibility with OpenSocial

TOC

This version of the Portable Contacts specification is currently wire-compatible with the overlapping portion of the OpenSocial RESTful Protocol version 0.8.1 [OpenSocial] (Panzer, J., “OpenSocial 0.8.1 RESTful Protocol Specification,” .). Specifically, *any compliant OpenSocial RESTful Protocol 0.8.1 Provider is also a compliant Portable Contacts Provider*, because they are specified to use the same Authorization methods (OAuth), Additional Path Information, Query Parameters, and Contact Schema. The OpenSocial and Portable Contacts communities chose to wire-align our respective specs in order to maximize widespread adoption of a single API for accessing people data. It is our intention to maintain this compatibility going forward, so long as it is feasible, and so long as the changes required are compatible with the Goals and Approach of this spec. Although Portable Contacts is an independent spec, with a more limited scope than OpenSocial, any proposed changes to either this Portable Contacts spec or the OpenSocial RESTful Protocol should be considered in the context of both communities, and we should strive not to break compatibility unless it is truly necessary, e.g. if the goals of the two communities diverge significantly in the future.

---

10. References

TOC

10.1. Normative References

TOC

[RFC2119]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[RFC2425]	Howes, T., Smith, M., and F. Dawson, “A MIME Content-Type for Directory Information,” RFC 2425, September 1998 (TXT, HTML, XML).
[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 (TXT, PS, PDF, HTML, XML).
[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 (TXT, HTML, XML).

10.2. Informative References

TOC

[OAuth Core 1.0]	OAuth, OCW., “OAuth Core 1.0.”
[OAuth Discovery]	Eran Hammer-Lahav, E., “OAuth Discovery 1.0.”
[OpenSearch]	Clinton, D., “OpenSearch 1.1.”
[OpenSocial]	Panzer, J., “OpenSocial 0.8.1 RESTful Protocol Specification.”
[XRDS-Simple]	Hammer-Lahav, E., “XRDS-Simple 1.0.”
[google-sgnodemapper]	Fitzpatrick, B., “Social Graph Node Mapper.”

Author's Address

TOC

	Joseph Smarr
	Plaxo
Email:	joseph@plaxo.com
URI:	http://josephsmarr.com/

Full Copyright Statement

TOC

Copyright © The IETF Trust (2008).
This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.
This document and the information contained herein are provided on an “AS IS” basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.
Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.
The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.