mailmaint
Internet-Draft
Intended status: Standards Track
Expires: 3 September 2026

L. Dusseault
Data Transfer Initiative
H. J. Happel
audriga
A. Melnikov
Isode Ltd
2 March 2026

Personal Data Portability Archive
draft-happel-mailmaint-pdparchive-02

Abstract

This document proposes the Personal Data Portability Archive format (PDPA), suitable for import/export, backup/restore, and data transfer scenarios for personal data.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://lisad.github.io/draft-happel-mailmaint-pdparchive/draft-happel-mailmaint-pdparchive.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-happel-mailmaint-pdparchive/.

Discussion of this document takes place on the mailmaint Working Group mailing list (mailto:mailmaint@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/mailmaint/. Subscribe at https://www.ietf.org/mailman/listinfo/mailmaint/.

Source for this draft and an issue tracker can be found at https://github.com/lisad/draft-happel-mailmaint-pdparchive.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

As part of communication protocols, the IETF has standardized a number of data formats such as the Internet Message Format [RFC5322], [vCard], [iCalendar], or, more recently, [JSContact] and [JSCalendar].

While mainly designed for interoperability, many of these data formats have also become popular for data portability, i.e., the import/export of data across different services. The growing importance of data portability however demands an open standard archive format which can deal with different types of personal data in a homogeneous fashion.

To this end, this document proposes the Personal Data Portability Archive format (PDPA), suitable for import/export, backup/restore, and data transfer scenarios for personal data. It is compatible with both IMAP and JMAP and should be suitable as an interchange format between related software and services such as for email, contacts, calendaring, tasks, or files.

The approach is to define JSON formats, folder structure, and a common compression format. Additional specifications will likely define a protocol how these files can be requested from, imported into, or transferred between servers, but this specification can be used as-is with user-directed imports or exports.

2. Conventions and Definitions

The term "personal data" refers to persistent data which users created and managed within applications or services. Classic examples are emails, contacts, calendars, tasks, notes, or files. Other examples might be fitness tracking records, energy bills, or location history.

The term "data portability" refers to the right or technical procedure to use or transfer personal data across different applications or services.

The terms "message" and "email message" refer to "electronic mail messages" or "emails" as specified in [RFC5322]. The term "Message User Agent" (MUA) denotes an email client application as per [RFC5598].

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Goals

The core goal is to provide an open and extensible archive and transfer format for personal data. This includes data types that are subject of existing IETF protocols, such as email and groupware data (e.g., contacts, calendars, tasks) but may also include further types of personal data.

Goals will be refined by use cases and cross-cutting technical goals in the following sub-sections.

JSON is used as a widely interoperable base format that systems can easily translate their internal data representation into. Wherever possible, fields, values and data structures are derived from existing IMAP/JMAP specifications to remain compatible with both.

3.1. Use cases

Many of these use cases benefit greatly from a simple, read-only export format, compared to having IMAP and CalDAV access to personal data. While these use cases may suggest interesting niche features, adding those may be left to future specs. For example, legal and forensics use cases might benefit from signing features, but nevertheless this specification would advance those use cases even without signing features.

3.1.1. Data portability

A main use case for the novel format is to allow exporting the full user data managed by services or software products into a simple file (or set of files) which is under full control of the user.

The user might use such export for backup, archiving, or for importing when switching to another service or software (i.e., migration).

Depending on the type of data, exporting/importing can be a time-consuming process. Particularly for the case of switching services, PDPA should allow to minimize the time period during which a user cannot use the origin system but also the destination system is not yet ready.

3.1.2. Read-only data access

General access to data powers all kinds of analysis and applications. Exporting content and metadata in a structure suitable for ease of use in data pipelines or analysis tools would make these much easier.

3.1.3. Incremental data access

Beyond snapshot backups/exports, the format should optionally allow for incremental data access - knowing what new data has been added.

* Automated maintenance of a permanent, incremental mirror of user data, e.g. for instant restore

* Compliance or audit logs

* Spam tracking and analysis

* CRM or productivity integration scenarios that are read-only

3.1.4. Synchronization

Data portability does not just allow users to switch from one service to another one, but to let users benefit from 3rd party services getting access to their data (at the request of the user). Simple synchronization features could make this much better.

For example, current online systems that allow importing contacts are not often suited to maintaining one's address book on two systems. Re-importing a contact into a system that already has that contact often results in duplicating the exact same contact, whether or not there have been edits, making repeated synchronization practically infeasible. It should be easy to do a significantly better job of this with some attention to object IDs and modification timestamps.

We however do not attempt to solve two-way synchronization via export files. It would require significant additional work to allow two systems, neither of which is agreed-upon to be the source of truth, to reliably synchronize changes from both. In comparison, solving one-way synchronization only requires agreed-upon usage of existing fields and values.

3.1.5. Dataset exchange

PDPA should be usable to exchange and share larger data sets than just one user, or to share a single user's data outside the context where the user knows what it is and where it came from.

Potential applications of this are:

* The ability to exchange test data and known mailstore state, e.g. for conformance testing or internal functional tests

* Legal discovery and forensics use cases may benefit from a standard export format, such that investigators can expect a great deal of consistency when collecting data from different systems.

* Researchers may be able to collect archive files through data donations and use as input to research.

3.1.6. Data persistence

The format MAY be used as a development-time active persistence layer for user data in, e.g., email clients or applications. It is not intended as, or suitable for, a production-level persistence layer.

3.2. Technical Goals

Besides actual use cases, there are a number of side requirements and goals for PDPA.

3.2.1. Email standards compatibility

Data formats should aim for compatibility with JMAP data formats for the sake of interoperability and synergies in software libraries.

Dedicated JMAP API methods for exporting and importing the format described here, or for related server-to-server transfer protocols are out of the scope of this document.

Due to its specifics and ubiquitous usage, the Internet Message Format [RFC5322]; latest revision of [RFC2822]/[RFC822] should be the core of representing individual email data.

This specification should ideally describe mappings between PDPA and existing mailbox persistence schemes such as Maildir or [MBOX].

3.2.2. Interoperability

It should be mostly possible to use personal data exports from one system with different software or services. When a source exports personal data it can include all the information it would need for a fully-functional import, _however_ destination systems running different software may not be able to import all of that information (especially if it includes non-standard features) and use it exactly the same way. This specification does not attempt to achieve perfect interoperability between diverse systems, but instead to make reasonable trade-offs.

3.2.3. Extensibility

This format should be extensible to accommodate types of personal data not explicitly mentioned or foreseen when writing these specs.

3.2.4. Flexible granularity

This format should allow flexible granularity in two ways:

1) It should enable easy access to separate types of data (e.g., emails vs. contacts), e.g. to allow for partial imports or exports

2) While ideally representable as a single file, archives may also span several files due to reasons such as file size restrictions or incremental generation logic.

(The ability for a user to export and/or backup an entire email account requires some accommodation of large amounts of data and risks of interruptions in downloads. Splitting exports into multiple files during export is one possible solution.)

3.2.5. Accessible for local tooling

PDPA should allow easy access for local tools (e.g., CLIs). While this may sound obvious, it is a key factor for the intended versatility of the format.

3.2.6. Efficiency

Since certain kinds of personal data might involve large quantities of data, major use cases for PDPA should be realizable in an efficient manner.

For now, this is stated as an abstract guiding principle. Its actual dimensions and trade-offs need to be refined while evolving this specification.

3.3. Related work

Many email server implementors have found it desirable to have one or more file formats for storing email in a file system even when the primary active email storage is more commonly a database. Examples include [PST] files (Outlook), NSF (Notes), [GoogleTakeout], Maildir, MBOX. File formats are already used for interoperability in many cases even when not standardized.

This specification follows that pattern in order to build on these partial successes. By standardizing one format, we expect to be able to satisfy use cases that are harder to satisfy with a plurality of formats, such as use cases for server-to-server transfer of email account data during account migrations. Specifications that explain how to create these archives in different situations can refer to this specification.

4. Approach

4.1. Using JSON (mostly)

JSON is used in this spec for new metadata and for objects including contacts, tasks, events and notes. However, the Email Message Format [RFC5322] is used for email message content. Individual items are stored in individual files, which are referenced in collection metadata. Finally these JSON and other file formats are packaged and compressed together in a standard but flexible way.

Our rationale for using JSON to the extent reasonable:

* We envision an export format being used not just by developers of full IMAP servers but also by developers building task management systems, calendar systems that don't include email, etc.

* We should minimize requiring multiple libraries to parse different formats. If the Metadata is going to be in JSON, it would really help to have the item data in JSON.

* Personal data formats must be extensible and extensibility for JSON is well-understood.

Using JSON for all the data _except_ EML files was carefully considered. EML files are rather specialized and more challenging to replace. Much of email is not structured data but content and involves MIME. EML is more likely to be a system's native data store, unlike VCARD and VTODO which are most commonly transformed for use in a relational data store for active use. Finally, email signature implementations like S/MIME and OpenPGP would be less disrupted by keeping EML.

Information not covered by these existing file formats, but still necessary for migration or backup/restore of an email account, is packaged into JSON files. JSON structure and values are defined with CDDL [RFC8610]. The JSON files for email folders and other collections contain references to individual resources by unique ID and filename.

4.2. Approach to partial updates

Our use case requirements above (use_cases) included some very common personal use cases that motivate exporting only a time-limited set of items, but retaining the ability to use that subset export to update a previously acquired or maintained snapshot. These are partial updates - partial exports able to update a full copy. Our approach is to define a version of the archive format that includes a subset of a repository's content, and additionally some change markers necessary to maintain consistency.

A partial-update export

* Contains new items just like a full export.

* Omits unchanged items from before the time cutoff.

* Does NOT list unchanged items in folder listings either.

* Allows updated items to be identified so they can replace previous versions

* Allows deleted items to be identified so they could be removed in the updated snapshot

The existing definitions for UIDs and 'updated' in JMAP and IMAP should make this quite possible. Also IMAP MODSEQs [RFC7162] can be used for flag changes.

4.3. Approach to synchronization

Email and calendaring services appear to already do synchronization just fine, but this is via a client-server model. The client drives the read and write requests until content is synchronized, using mostly UIDs and timestamps.

Archive and export formats aren't part of this client-server model, and the work to make server-to-server or peer-to-peer synchronization work perfectly is substantial (involving features such as version numbers or change logs -- features that aren't commonly standardized for the objects handled in this spec). Still, there are some things possible with archive formats and the current object definitions that are sensible. Our approach is to describe what is possible with the fields that exist, and mandate those fields be used, so that users aren't left with multiple locations for their data and no way to repeatedly synchronize them.

As an illustrative use case, let the user have contacts stored in one email service and also in one mobile device platform doing online backups. The email platform creates contacts when the user emails new recipients, or receives contact information over email. The mobile device platform synchronizes contact information from a phone. The email service is not a client of the mobile device platform, nor is the mobile device platform a client of the email service, so client-to-server protocols cannot directly synchronize the data between these two services. A user attempting to solve this by repeatedly importing contacts into one system from the other may find this works poorly - for example, it might create new contact objects over and over for the same contact data even if it is unchanged, and deleted objects may re-animate after being re-copied.

Both servers ought to allow exporting of contact data (along with any other data covered in this specification), including especially the UID and updated (timestamp) fields. This would allow at least some sensible personal workflows or for 3rd party tools to make synchronization work better.

When importing contact data (along with any other data covered in this specification), a service needs to take note of the UID in the import, and use it as the UID for creating a new object, so that it may be later avoid being recreated. If the service importing already has the said UID, the service should compare 'updated' timestamps and use that to decide how to update the object with new values. An object that hasn't changed since last imported should remain unchanged and not updated.

This approach to synchronization is imperfect. Let the user have performed one synchronization by exporting from the email service and importing to the mobile device platform. If the user updates a contact on the email service (e.g adding a street address) and the mobile platform also updates the contact (e.g. adding an avatar link), then the user attempts to repeat the synchronization, both services will have a new 'updated' timestamp on the same object UID, and the earlier of the two changes will get wiped out. Nevertheless, a disciplined user can remember to only make changes on the email service and always copy them over to the mobile platform, and avoid many such lost updates.

Based on the approach described here, this specification standardizes some behavior for both exporters and importers, to maximize potential success.

5. Solution Requirements

These technical requirements on the solution are intended to meet the goals above, and to add more specifics about how those goals are intended to be met with the architecture chosen. This section does not attempt to translate the solution requirements into implementor requirements.

Folder format requirements

* can include partial results, showing only a subset of objects within the folder

* can include a human-readable representation of the meaning of that subset (e.g. a date filter, or recipient filter)

Email object format requirements

* can maintain full fidelity including preserving character sets, content transfer encoding of body parts and exact MIME structure

Compression and packaging of resources

* Servers can bundle resources together in different ways to be flexible in handling size and network limitations. Servers must be able to choose optimal file size and organization of information within files.

Synchronization requirements

* Can see which items are identical in two systems, e.g. one system previously imported an item exported by the other.

* Can detect changes made since the last time an item was imported, in a way that supports replacing an older version previously synchronized with a newer version that has edits.

6. File format

This section describes the internal "raw" file format of a personal data portability archive. For discussion about a surrounding container format, see section "open issues".

PDPA in general consists of:

* A main metadata file ("index.json")

* Top-level folders for each data type ("/mail")

* Subfolders representing actual collections of individual data items plus additional metadata files

6.1. Index file(s)

The index.json file consists of three main sections:

* archive: general information about the archive

* dataset: characteristics of the dataset itself

* datasource: meta-information about the dataset

The following sections propose some initial properties which are still subject to discussion.

6.1.1. Archive section

Key	Description	Example value
id	Archive identifier	123
name	Human readble label	Jane’s data export (2025-10-19)
note	Note	Personal account export
legal	Legal desclaimer	Private data
timestamp	Archive timestamp	2025-10-19-18-00
version	PDPA spec version	PDPA v1.0
generator	Archive generator	PDPA exporter v0.9

Table 1

6.1.2. Dataset section

Key	Description	Example value
extent	Extent of the archive (full, partial)	FULL
selector	Select critia for partial datasets (date, folder, size, custom)	NONE
datatypes	List of data types	MAIL
langaguetag	BCP 47 language tag for the dominant language in the dataset	en-ca
timezone	IANA tz identifier for the dataset	"America/ Montreal"

Table 2

6.1.3. Datasource section

Key	Description	Example value
service	Information about the source service (id, url, ..)	TBD
account	Information about the source account (id, type, ...)	TBD

Table 3

6.2. Folder structure

* Folders can be nested. Any kind of content here can be within nested folders -- this is a necessary feature for email, but extends to content like contacts that aren’t always represented in nested folders. (TODO: elsewhere, describe the requirements for an importing system to preserve folders or not.)

* Names of files do not have to be globally unique. Indexes and folder contents listings can name files relatively to their location in the archive structure, which means that references may not be resolvable if that context is lost.

* Individual content items are individual files. This may not always be the easiest choice for exporters who must generate a large number of files for individually small items (contrast to a JSON stream including all objects) but as an archive format, the individual files allow more clarity in individual handling, transactions and errors.

```
   index.json
   /mail/
       /Archive/
           folder.json
           m1.eml
           m2.eml
           m3.eml
           ...
       /Archive/2023/
           folder.json
           m1.eml
           m2.eml
           m3.eml
           ...
       /Archive/2024/
           folder.json
           m1.eml
           m2.eml
           m3.eml
           ..
       /INBOX/
           folder.json
           m1.eml
           m2.eml
           m3.eml
           ...
       /Sent Mail/
           folder.json
           m1.eml
           m2.eml
           ...
   /contacts/
        contact1.json
        contact2.json
        ...
   /calendars/
       /calendar2/
           event1.json
           event2.json
   /sieve/
   /blob/
       ...?
```

6.2.1. File and folder names

Because filenames can be generated by the exporting server, it is always possible to generate non-colliding filenames. Display names are NOT intended to be determined by file names, but instead by fields within each file. Similarly, filenames are NOT intended to be globally unique IDs.

Mail folder names are defined in [RFC9051] (IMAP v4 rev2) with great freedom for servers. Servers may or may not treat mailbox names as case sensitive. Folder names may even include non-graphic characters, "%" and "*". Hierarchy separators may even differ among IMAP servers although "/" is probably most common.

Since this specification is new, it is possible to be more constrained. This specification only supports "/" as a folder separator.

6.3. Data formats

6.3.1. Email

Each IMAP/JMAP folder is represented as subdirectory under "mail" directory. For example, the folder INBOX would be represented as "mail/INBOX", and the folder "Archive/2024/2024-12" would be represented as "mail/Archive/2024/2024-12". Folder names are encoded in UTF-8.

TODO: how to signal removal of a folder in an incremental archive? Need to add some kind of tombstone mechanism.

Each folder metatadata is described by "folder.json", which has the following format:

Attribute Name	Type	Mandatory?	Comment
allowed_keywords	array of strings (IMAP keywords)	No	PERMANENTFLAGS minus "*" [IMAP4]
last_uid	unsigned 32 bit integer	Yes	Last UID assigned in the folder. It is UIDNEXT value minus 1 [IMAP4]
recent_uid	unsigned 32	No	Lowest UID of a

	bit integer		message with the \Recent flag [IMAP4]
uidvalidity	unsigned 32 bit integer	Yes	UIDVALIDITY value [IMAP4]
is_subscribed	boolean	Yes	Is the folder returned by IMAP LSUB? [IMAP4]
myrights	string	No	See Section 3.5 of [RFC4314]. For example "rwiptsldaex"
highest_modseq	unsigned 64 bit integer	No	HIGHESTMODSEQ value [RFC7162]
special_use	string	No	[RFC6154] SPECIAL-USE value. E.g. "inbox", "sent", "drafts", "junk", etc.
sort_order		No	See Section 2, [JMAP]]
uids	map from UIDs to strings (relative filenames of messages)	Yes	Mapping from UIDs to corresponding message files included in the archive
flags	map from UIDs to array of strings	Yes	IMAP flags assigned to the message, excluding \Recent
modseqs	map from UIDs to unsigned 64 bit integers (modsequences)	No	Per message MODSEQ values [RFC7162]

original_name	string	No	Original folder name (relative to its parent, if any) if the name can’t be represented in filesystem, e.g. if it includes special characters
comment	string	No	Can include information about partial export or filter used in human readable UTF-8 text
removed	array of unsigned 32 bit integers	No	List of messages (UIDs) removed since the last export

Table 4

CDDL description of "folder.json" is included below:

```
;; /// Or possibly use ranges for 2 types below?
u32 = uint .size 4
u64 = uint .size 8

; uid => message filename map
uid_map = {* uid => filename}
uid = u32
filename = tstr

flags_map = {* uid => flags}
flags = [tstr]

modseq_map = {* uid => modseqs}
modseqs = u64


folder_info = {
```

```
  ? allowed_keywords: [tstr],
  last_uid: u32,
  ? highest_modseq: u64,
  ? recent_uid: u32,
  uidvalidity: u32,

  is_subscribed: bool,

  ; See RFC 6154 (SPECIAL-USE). E.g. "inbox", "sent", "drafts", "junk", etc.
  ? role: tstr,

  ; See Section 2, RFC 8621.
  ? sort_order: u32,



  ; See Section 3.5 of RFC 4314. For example "rwiptsldaex"
  ? myrights: tstr,

  ; Maps UIDs to the corresponding relative file names (names of .eml files)
  uids: uid_map,

  ; Maps UIDs to the corresponding array of flags/keywords (as strings)
  flags: flags_map,

  ; Maps UIDs to the corresponding modseqs (u64). See RFC 7162.
  ? modseqs: modseq_map,

  ; Original folder name if the name can't be represented in filesystem
  ? original_name: tstr,

  ; Can include information about partial export or filter used
  ; in human readable UTF-8 text
  ? comment: tstr,

  ;; The following attributes are only used for incremental exports:
  ? removed: [u32],
}
```

EML (.eml) file for each message, in order to avoid reconstructing them. Names of EML files are referenced from the "folder.json" file.

Example of folder.json (full export):

```
{
  "name": "AVClub",
  "allowed_keywords": ["$forwarded", "$MDNSent", "$ismailinglist"],
  "last_uid": 16,
  "highest_modseq": 6371729,
  "recent_uid": 15,
  "uidvalidity": 1107190787,
  "is_subscribed": true,
  "special_use": "sent",
  "sort_order": 1,
  "myrights": "rwiptsldaex",

  "uids": {
     "1": "msg-1.eml",
     "3": "msg-3.eml",
     "15": "imported-ABC.eml"
  },

  "flags": {
     "1": ["$seen"],
     "3": ["$seen", "$flagged"],
     "15": ["$answered", "$forwarded"]
  }
}
```

Figure 1: A basic folder.json example

Issue: Are UIDs compared as strings or integers? JSON requires UIDs to be strings if they’re used as field names. How about when we’re using "last_uid" or "recent_uid"?

Example of folder.json that shows incremental changes from the previous export shown above. In this example 2 messages with UIDs 3 and 15 were removed. Message with UID 1 has updated flags. Several new messages were added, some of them are with flags set.

```
{
  "allowed_keywords": ["$forwarded", "$MDNSent", "$ismailinglist"],
  "last_uid": "21",
  "highest_modseq": 6371845,
  "recent_uid": "20",
  "uidvalidity": 1107190787,
  "is_subscribed": true,
  "role": "sent",
  "sort_order": 1,
  "myrights": "rwiptsldaex",

  "uids": {
     "1": "msg-1.eml",
     "17": "msg-17.eml",
     "19": "msg-19.eml",
     "20": "20.eml",
     "21": "21.eml"
  },

  "flags": {
     "1": ["$seen", "$answered"],
     "17": ["$seen", "$answered", "$forwarded"],
     "19": ["$seen"],
     "20": [],
     "21": []
  },

  "removed": ["3", "15"]
}
```

Figure 2: A folder.json example with incremental changes

6.3.2. Contacts

[vCard] has long been the basis for address book and contact data representation in structured data files. The specifications for [JSContact] and JMAP for Contacts [RFC9610] do a bunch of the work to explain how to do this in JSON, and in particular RFC9610 explains how to express references between objects (e.g. an address book and a contact in that address book) which is useful for a full export that can have its references reconstructed. This section explains how to use the fields and structures of those specifications within a PDP Archive export.

6.3.2.1. Individual Contact Items

Individual contact items build on [JSContact] which builds on [vCard].

* The globally unique uid property is mandatory in JSContact and MUST be included in PDP archive.

* The updated property is optional in JSContact but MUST be included in PDP archive.

* The rev property defined in [vCard], which is not included in [JSContact], may already be available in implementations. It MAY also be included as a field on a contact, in which case it is a simple value field holding a timestamp.

* The @type property should be "ContactCard". Note that [JSContact] uses a value of "Card" for @type and registers that in https://www.iana.org/assignments/jscontact/jscontact.xhtml, but JMAP for Contacts uses "ContactCard" and registers that in https://www.iana.org/assignments/jmap/jmap.xhtml.

We make some specific requirements on the updated value so that it can be useful for synchronization. See the section on updated and uid specifically Section 6.4.1.

When the structured data is prepared, a contact can be exported in a file with an arbitrary name using a limited set of characters suitable for interoperability across filesystems.

For example, a file called ’contact1.json’ could contain:

```
{
   "@type": "ContactCard",
   "version": "1.0",
   "uid": "22B2C7DF-9120-4969-8460-05956FE6B065",
   "id": "42",
   "updated": "2021-10-31T22:27:10Z",
   "kind": "individual",
   "addressBookIds": [
      "062adcfa-105d-455c-bc60-6db68b69c3f3"
   ],
   "name": {
       "components": [
         { "kind": "given", "value": "John" },
         { "kind": "surname", "value": "Doe" }
       ],
       "isOrdered": true
   },

   "relatedTo": {
      "urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6": {
        "relation": {
          "friend": true
        }
       }
   },

   "notes": {
     "n1": {
       "note": "Open office hours are 1600 to 1715 EST, Mon-Fri",
       "created": "2022-11-23T15:01:32Z",
       "author": {
         "name": "John"
       }
     }
   }
}
```

Figure 3: A contact.json example

For clarity, this example includes: * How a card can reference address books which are exported as separate files in the overall export * How a card can reference other cards using relatedTo * A card can contain arbitrary notes - those are not necessarily exported as separate files even though notes are also an object that can be included as individual files in a PDPArchive export.

TODO: figure out if these can have RFC9610 "id" field

Because a ContactCard item can reference an AddressBook item, if a system exports contacts belonging to address books it SHOULD also export the referenced AddressBook objects. Likewise, it SHOULD export the other ContactCard objects that are referenced in the 'relatedTo' field. A permission or scope inconsistency would be one reason why the exporting system would not do so. For example, if the user chose to export only a public address book containing the "John Doe" contact, and not the private "Wedding guests" address book that John Doe also belonged to, then the private address book would either appear as an unresolvable ID or be cleaned up so that it didn't appear (implementor's choice). Likewise, when "John Doe" is exported as part of a single address book export, but the friend relation in relatedTo is not exported because they're not in the same address book, the relatedTo value may be included in the export even if not resolvable by some users of the export file.

6.3.2.2. Group Contact Items

Group contact items also refer to other contact items. A file with an arbitrary name like "contact2.json" could include:

```
{
   "@type": "ContactCard",
   "kind": "group",
   "name": {
     "full": "The Doe family"
   },
   "uid": "urn:uuid:ab4310aa-fa43-11e9-8f0b-362b9e155667",
   "updated": "2021-10-31T22:27:10Z",
   "members": {
     "urn:uuid:03a0e51f-d1aa-4385-8a53-e29025acd8af": true,
     "urn:uuid:b8767877-b4a1-4c70-9acc-505d3819e519": true
   }
}
```

Figure 4: A group contact file example

As with individual ContactCard items referencing objects that are not exported at the same time, a group contact can contain references that are not resolvable within the export. If the user chooses to export all address books then presumably the "The Doe family" group members can all be found somewhere in the export, but if they export only the address book containing "The Doe family" group and not the address books containing individual members, those IDs would not be found in the export.

6.3.3. Using RFC9610 address book objects

The [vCard] specifications never defined a representation for address books. Nor did [JSContact]. JMAP for Contacts [RFC9610] does. Its model is clearly that of non-exclusive collection membership: a Contact item may appear with the same UID in multiple Address Books, and if the Contact item with that UID is updated in one it is updated in the other also.

Individual address book objects are returned in JMAP protocol messages with protocol wrappers. It is the items inside the "list" element inside "AddressBook/get" that are nearly ready to be represented as individual files in a PDPArchive. However, some things are missing: * uid is called ’id’ in JMAP for Contacts but this specification REQUIRES uid. * updated is required * The @type of AddressBook should be included within the data

This example copies the examples in [RFC9610] so that interoperability between this spec and that one is clear.

A file with an arbitrary name like address-book1.json would contain:

```
{
   "@type": "AddressBook",
   "uid": "062adcfa-105d-455c-bc60-6db68b69c3f3",
   "updated": "2020-01-09T14:32:01Z",
   "name": "Personal",
   "description": null,
   "sortOrder": 0,
   "isDefault": true,
   "isSubscribed": true,
   "shareWith": {
     "3f1502e0-63fe-4335-9ff3-e739c188f5dd": {
       "mayRead": true,
       "mayWrite": false,
       "mayShare": false,
       "mayDelete": false
     }
   },
   "myRights": {
     "mayRead": true,
     "mayWrite": true,
     "mayShare": true,
     "mayDelete": false
   }
}
```

Figure 5: An address book file example

address-book2.json would contain:

address-book2.json

```
{
   "@type": "AddressBook",
   "uid": "cd40089d-35f9-4fd7-980b-ba3a9f1d74fe",
   "updated": "2020-01-09T14:32:01Z",
   "name": "Autosaved",
   "description": null,
   "sortOrder": 1,
   "isDefault": false,
   "isSubscribed": true,
   "shareWith": null,
   "myRights": {
     "mayRead": true,
     "mayWrite": true,
     "mayShare": true,
     "mayDelete": false
   }
}
```

Figure 6: Another address book file example

Note that the first example includes a shareWith value, showing that the user's AddressBook has been shared with in this case one other principal with the id "3f1502e0-63fe-4335-9ff3-e739c188f5dd". This information can be exported and may be quite useful in case of backup/restore use cases. However, it may not be useful in other administrative domains where the same concept of principals does not allow the Principal ID to be resolved against the correct account. In any case, the object referred to by this Principal ID is not itself given representation in the PDP Archive export.

6.3.4. Calendar events, tasks and groups

[JSCalendar] is the basis for representing events, tasks and groups in JSON. This section explains how to export individual events and tasks within an archive. JMAP for Calendars (https://datatracker.ietf.org/doc/draft-ietf-jmap-calendars/) does provide some additional considerations when producing calendar data from a JMAP system or to be consumed by a JMAP system, so it is also a normative reference.

Note on [CalDAV] compatibility: Although CalDAV servers are fairly common, they support the older VEVENT and VTODO syntax. This specification requires the JSCalendar syntax instead. Either way, a server building a personal data archive is likely transforming an internal implementation-specific relational data format to an export format.

Note on ETag: CalDAV servers use the event's UID to identify the same object, and use ETags to identify changed events, so that a CalDAV client may make sure it has the version a server has before it updates an item, solving the lost-update problem. Since this specification doesn't attempt to solve the lost-update problem as well as client-server protocols can, and since JSCalendar does not include the ETag of a calendar event in any way, this specification does not include any requirements for ETags.

Notes on specific fields:

* The globally unique uid property is mandatory in JSCalendar and MUST be included. See JMAP Calendars draft-ietf-jmap-calendars-26 section 1.4.1 for when the uid property can appear the same for multiple recurrences of the same underlying event.

* The updated property is mandatory in JSCalendar and MUST be included.

* The sequence value is optional in JSCalendar but SHOULD be included if available.

* The @type property for one of these items MUST be "Event", "Task" or "Group".

* Recurrence rules SHOULD be fully exported, unless it's clear from the use case or user request that the destination for the data wants expanded recurrences within a specific time period.

* The calendarIds field defined in JMAP Calendars is REQUIRED in order to match up events to the calendar they are supposed to appear in.

For example, a file called event1.json could contain:

```
{
  "@type": "Event",
  "uid": "2a358cee-6489-4f14-a57f-c104db4dc2f2",
  "updated": "2020-01-09T14:32:01Z",
  "title": "Board Meeting",
  "start": "2024-10-25T09:00:00",
  "timeZone": "Europe/London",
  "duration": "PT1H30M",
  "participants": {
    "1": {
      "@type": "Participant",
      "name": "Jane Doe",
      "sendTo": {
        "mailto": "jane@example.com"
      },
      "roles": {
        "attendee": true
      }
    }
  },
  "calendarIds": {
    "062adcfa-105d-455c-bc60-6db68b69c3f3": true
  }
}
```

Figure 7: Event example

The event object includes a calendarIds property, which links it to the calendar collection it belongs to.

6.3.5. Calendar Collection Items

Calendar collection items are built using JMAP for Calendars (draft-ietf-jmap-calendars-26).

If a system exports events belonging to calendars, it SHOULD also export the referenced Calendar objects.

A file with an arbitrary name, such as calendar1.json, in a directory (e.g., \calendars\calendar2) would contain the calendar’s metadata:

```
{
  "@type": "Calendar",
  "uid": "062adcfa-105d-455c-bc60-6db68b69c3f3",
  "updated": "2020-01-09T14:32:01Z",
  "name": "Work Calendar",
  "color": "#123456",
  "sortOrder": 0,
  "isDefault": true,
  "isSubscribed": true,
  "myRights": {
    "mayRead": true,
    "mayWrite": true,
    "mayShare": true,
    "mayDelete": false
  }
}
```

Figure 8: Calendar example

The uid value here corresponds to the ID used in the calendarIds property of the individual event item.

6.3.5.1. Tasks

Tasks are also defined by [JSCalendar] using the "Task" object type. As with events, tasks MUST include the uid and updated fields to support synchronization.

For example, a file called task1.json could contain:

```
{
  "@type": "Task",
  "uid": "7b0f69a6-6e3e-4f1b-85d8-c89b43d2f2a1",
  "updated": "2022-11-23T15:01:32Z",
  "title": "Submit Quarterly Report",
  "status": "in-progress",
  "priority": 1,
  "due": "2024-12-31T23:59:59Z"
}
```

Figure 9: Task example

6.3.6. Notes

* VJournal is first defined in [iCalendar].

* VJournal also used in [CalDAV].

* However, they are NOT used in https://jmap.io/spec-calendars.html

Do we even have a JSON format for notes defined?

6.3.7. Files

6.3.8. Other

(LMD Note: I think this might better fit in an out of scope section - I think out of scope sections are useful for statements that explain why scope is limited. that's assuming we all agree that groups, freebusy and timezones are left out.)

Groups as defined in JSCalendar are NOT part of this archive format. Groups in JSCalendar can combine events and tasks in a container. This specification, for consistency and simplicity, uses folders and requires individual objects to be in separate files.

VFREEBUSY objects are not part of this archive format. Calendar software can calculate freebusy time from event data. Use cases that are not satisfied by this limitation could extend this archive format but understanding what it means to backup, restore, export or import freebusy data would need to be fleshed out.

VTIMEZONE objects are not part of this archive format. Timezones are more likely to be system objects referred to by calendar objects in modern calendar systems, than objects which are exchanged across systems.

6.4. Synchronization requirements

This section describes the requirements to achieve repeated one-way synchronization via export/import operations between software by different vendors. While limited, this still provides better functionality than what many end-users experience with their groupware software and services today.

Supporting _repeated_ synchronization means that the export from system A and import to system B can happen over and over again without needlessly duplicating items. Supporting _one-way_ synchronization means that changes in the system with the exporter role propagate reliably to the system with the importer role, but not in the reverse direction. Some of the constraints here arise from the fact that the two systems may not directly connect, and the import may be time-delayed from the export.

This limited solution for export/import sync may also be used for more direct system-to-system transfers such as service-to-service data transfers, repeated data access requests or data migrations, although some of those use cases could be solved much better with direct negotiation of features.

6.4.1. Always include 'uid' and 'updated'

These requirements apply to [JSContact], JSTask and [JSCalendar] objects when exported or imported using the formats in this specification, because these all have 'uid' and 'updated' values.

Requirements:

* exporters MUST include the UID and updated fields.

* The 'updated' field MUST be exported in UTC and interpreted in UTC. Accurate system time is important.

* importers MUST use the UID in an imported object, if the importer is creating a new object, rather than invent a new UID.

* importers MUST search for existing objects with the same UID, and if the object in storage is _similar enough_ (see Note below) to the import data, the importer SHOULD NOT change the object and MUST NOT update its 'updated' timestamp.

Recommendations:

* Importers SHOULD use caution with fields that are system-updated, especially frequently updated. Such fields SHOULD NOT change the value of 'updated' that is exported or used to decide whether to update an object during an import operation. See note below on 'updated'.

* Importers SHOULD apply common sense in updating internal or implementation-specific fields. This specification does not require the importer to include, omit, handle or disregard values for fields that it believes are internally-generated or implementation-specific. For example, a system in the role of exporter might export an event object with a video-conference room ID in a custom field. It can decide that it is sensible to export that value as a URL for external use. Later, the same system or one with code written for compatibility could import that event with the video-conference URL, and it would be sensible to avoid overwriting its own knowledge of the room ID with the URL.

* When importing a _changed_ or _new_ object with a UID and 'updated' value, the importer SHOULD set the 'updated' value to the one imported. Thus, if a Contact is updated on Jan 1, exported on Jan 2 and imported on Jan 3, the new or updated imported contact would show an 'updated' value of Jan 1.

Note on _similar enough_: This specification requires nuance in order to allow both reasonably consistent synchronization and reasonable behavior in a wide variety of use cases and implementations. The language above is intended to give implementors both guidance and wiggle room. For example, the importer could convert a DTSTART time from UTC to the user's local time and save it as the displayed start time. Later, re-importing the same object with the same UID, the importing code could be smart enough to realize that the time hasn't _actually_ changed, and avoid changing the 'updated' timestamp or creating a conflicting event. This logic could be implemented by saving separate fields (imported time vs display time), by keeping a log of updates (log entry stating that the system auto-converted start time from X to Y), or by other clever algorithms. Thus, the clever implementation can avoid the appearance of an object that changes every time the calendar is synchronized.

Note on _updated_: The definition of 'updated' in [JSContact] is not rigorous or nuanced. "when the data in the Card was last modified" could refer to several instances of the card -- its internal implementation, its representation in an email share, its representation in an HTTP GET response ([CardDAV]). It's not specified whether 'updated' is the same as REV in [vCard], which is defined differently. Neither definition explicitly covers vendor-specific fields. Thus, this specification makes additional recommendations for handling 'updated':

* The value of 'updated' SHOULD only change when two conditions hold: the end-user makes a decision to change a value of a user-visible field, AND the export of the JSContact shows a different value.

* Thus, non-user-visible fields like 'version' could be changed without causing the 'updated' value to change. A value such as 'language' could be set without changing 'updated' (if an implementation infers the language tag and begins to include 'fr-CA' as the language value in exports instead of no language, nevertheless this doesn't change the user-visible content).

* If implementations need to manage the synchronization of vendor-specific fields, a vendor-specific field like 'example.com:updated' can be used rather than affect the user-visible synchronization made possible by 'updated'.

Implementations could possibly also handle ’updated’ differently when used for export/import using the formats in this specification, than when the same field is handled in other code paths.

We recognize that this understanding of ’updated’ is highly judgement-dependent. The same field can change in one way and cause a change to ’updated’ and in another way may not (the example of server inferring the language is ’fr-CA’ vs the user explicitly setting it). It is likely to be frustrating to protocol designers and implementors (as it is to the authors of this specification) that the definition is so wobbly. We’d love to know of better solutions that work with the status quo.

6.4.1.1. Synchronizing from and to CalDAV servers

[CalDAV] uses URLs, ETags and UIDs for synchronizing changes between two systems reliably, but it relies upon client-server architecture, where the server is the "source of truth" and the client must manage its local history and decide which things to update from the server and which things to tell the server to update. If a user is setting up synchronization or an implementor is building a system that involves synchronization, it may be best to use CalDAV if that is a feasible solution.

Nevertheless, we believe some of the use cases in our use case section (use_cases) motivate not only including calendar data in these archives for backup purposes, but also for partial updates. This works the same way it does for JSContact and JSCard objects.

6.4.2. Synchronizing address books

Build on [RFC9610]

6.4.3. Synchronizing mailbox folders

Because servers may differ in which characters they support in folder names, how many levels deep folders may be created, and even in what separator character is used to indicate folder hierarchy, difficulties in synchronizing folder names will definitely arise. Folder names that are not likely to be widely supported in other systems should be translated for export, because if the exporting system has a consistent translation algorithm, then even if the mailbox name looks different in the importing system it will still be imported consistently.

Systems that support mailbox IDs MUST include them in exports. Systems that do not (though it's strongly encouraged) SHOULD use the full mailbox name as the unique identifier value.

TODO: Also it would be good to include a "display name" in case the server has had to translate the mailbox name for compatibility. E.g. a server that has a mailbox named "%L33T%", but knows the "%" should not be exported because many servers forbid the "%", would translate the name consistently to _pc_L33T_pc_ or another set of safe characters and include a display name of "%L33T%" for reference and debugging.

6.4.4. Blobs and files?

Reference [RFC9404]?

7. Open issues

7.1. Container format

This document leverages existing data formats and adds certain files for representing metadata. While one may work with this raw data, most import/export scenarios will rather require the bundling of individual data items into one or few container files.

This document does not strive to invent its own container format, but may refer to existing ones.

High level options would be:

* Recommend using a container format without preferring a particular one
* Mandating a specific format
* ...?

Actual container formats likely differ in various dimensions:

* Ease of adding incremental data
* Ease of updating existing data
* Ease of accessing files
* Support for compression
* Support for data streaming

* Availability of library/tool support across platforms

* Internal file references

* Open standard

* ...?

Candidates

* tar/gz

* zip

* 7z

* zpaq

* ...?

See https://github.com/hhappel/draft-happel-mailmaint-pdparchive/issues/13

7.2. Encryption

Support for encryption of any kind is so far no requirement in the draft. However, an increasing number of services offers forms of data encryption. Implications for this draft may be considered.

"Encryption" might refer to various aspects:

* Existing encryption of individual files in the export

* Encrypting the complete export (incl. metadata?)

* ...?

See https://github.com/hhappel/draft-happel-mailmaint-pdparchive/issues/14

8. Implementation status

< RFC Editor: before publication please remove this section and the reference to [RFC7942] >

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942].

The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

9. Security Considerations

TODO Security

10. Privacy considerations

tbd.

11. IANA Considerations

11.1. File extension

Register .pdpa?

12. References

12.1. Normative References

[CardDAV] Daboo, C., "CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV)", RFC 6352, DOI 10.17487/RFC6352, August 2011, <https://www.rfc-editor.org/rfc/rfc6352>.

[JMAP] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP) for Mail", RFC 8621, DOI 10.17487/RFC8621, August 2019, <https://www.rfc-editor.org/rfc/rfc8621>.

[JSCalendar]
Jenkins, N. and R. Stepanek, "JSCalendar: A JSON Representation of Calendar Data", RFC 8984, DOI 10.17487/RFC8984, July 2021, <https://www.rfc-editor.org/rfc/rfc8984>.

[JSContact]
Stepanek, R. and M. Loffredo, "JSContact: A JSON Representation of Contact Data", RFC 9553, DOI 10.17487/RFC9553, May 2024, <https://www.rfc-editor.org/rfc/rfc9553>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/rfc/rfc5322>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9051] Melnikov, A., Ed. and B. Leiba, Ed., "Internet Message Access Protocol (IMAP) - Version 4rev2", RFC 9051, DOI 10.17487/RFC9051, August 2021, <https://www.rfc-editor.org/rfc/rfc9051>.

[RFC9610] Jenkins, N., Ed., "JSON Meta Application Protocol (JMAP) for Contacts", RFC 9610, DOI 10.17487/RFC9610, December 2024, <https://www.rfc-editor.org/rfc/rfc9610>.

12.2. Informative References

[CalDAV] Daboo, C., Desruisseaux, B., and L. Dusseault, "Calendaring Extensions to WebDAV (CalDAV)", RFC 4791, DOI 10.17487/RFC4791, March 2007, <https://www.rfc-editor.org/rfc/rfc4791>.

[GoogleTakeout]
Google, "Google Takeout", <https://takeout.google.com/settings/takeout>.

[iCalendar] Desruisseaux, B., Ed., "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 5545, DOI 10.17487/RFC5545, September 2009, <https://www.rfc-editor.org/rfc/rfc5545>.

[IMAP4] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003, <https://www.rfc-editor.org/rfc/rfc3501>.

[MBOX] Hall, E., "The application/mbox Media Type", RFC 4155, DOI 10.17487/RFC4155, September 2005, <https://www.rfc-editor.org/rfc/rfc4155>.

[PST] Microsoft, "[MS-PST]: Outlook Personal Folders (.pst) File Format", <https://learn.microsoft.com/en-us/openspecs/office_file_formats/ms-pst/141923d5-15ab-4ef1-a524-6dce75aae546>.

[RFC2822] Resnick, P., Ed., "Internet Message Format", RFC 2822, DOI 10.17487/RFC2822, April 2001, <https://www.rfc-editor.org/rfc/rfc2822>.

[RFC4314] Melnikov, A., "IMAP4 Access Control List (ACL) Extension", RFC 4314, DOI 10.17487/RFC4314, December 2005, <https://www.rfc-editor.org/rfc/rfc4314>.

[RFC5598] Crocker, D., "Internet Mail Architecture", RFC 5598, DOI 10.17487/RFC5598, July 2009, <https://www.rfc-editor.org/rfc/rfc5598>.

[RFC6154] Leiba, B. and J. Nicolson, "IMAP LIST Extension for Special-Use Mailboxes", RFC 6154, DOI 10.17487/RFC6154, March 2011, <https://www.rfc-editor.org/rfc/rfc6154>.

[RFC7162] Melnikov, A. and D. Cridland, "IMAP Extensions: Quick Flag Changes Resynchronization (CONDSTORE) and Quick Mailbox Resynchronization (QRESYNC)", RFC 7162, DOI 10.17487/RFC7162, May 2014, <https://www.rfc-editor.org/rfc/rfc7162>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[RFC822] Crocker, D., "STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES", STD 11, RFC 822, DOI 10.17487/RFC0822, August 1982, <https://www.rfc-editor.org/rfc/rfc822>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[RFC9404] Gondwana, B., Ed., "JSON Meta Application Protocol (JMAP) Blob Management Extension", RFC 9404, DOI 10.17487/RFC9404, August 2023, <https://www.rfc-editor.org/rfc/rfc9404>.

[vCard] Perreault, S., "vCard Format Specification", RFC 6350, DOI 10.17487/RFC6350, August 2011, <https://www.rfc-editor.org/rfc/rfc6350>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Lisa Dusseault
Data Transfer Initiative
Email: lisa@dtinit.org

Hans-Joerg Happel
audriga
Email: hans-joerg@audriga.com

Alexey Melnikov
Isode Ltd
Email: Alexey.Melnikov@isode.com