Document Upload API Design - Draft

Vets.gov
Patrick Vinograd - Ad Hoc LLC

Draft - Version 0.3.0 - 2018-04-24

Background

This document describes the design and rationale for the document upload API. This API is being provided as a proof of concept of a general-purpose VA API gateway. The use case is to allow authorized third-party developers to submit VBA documents and have a high degree of assurance that those documents will be processed and stored in the appropriate VA data system.

The Document Upload API passes data through to ICMHS aka “Central Mail API”.

Central Mail API accepts a payload consisting of a document in PDF format, zero or more optional attachments in PDF format, and some JSON metadata. The metadata describes the document, attachments, and identifies the person that the document relates to. This payload is encoded as multipart/form-data. A unique identifier supplied with the payload can be used to subsequently request the processing status of the uploaded document package.

Design

Authorization

API requests are authorized by means of a symmetric API token, provided in an HTTP header with name "apikey".

API Response Format

All JSON responses generated by this API are formatted according to the JSON API specification (http://jsonapi.org/).

- Successful responses have top-level keys "id" and "type", and the response-specific data fields are contained in a nested "attributes" JSON object. (http://jsonapi.org/format/#document-resource-objects)
- Error responses are returned as a nested array under top-level key "errors", though this API will only return a single error in any given response. (http://jsonapi.org/format/#error-objects)

Upload Operation

Allows a client to upload a document package (form + attachments + metadata).

1. Client Request: POST https://api.vets.gov/services/vba_documents/v0/upload
    - No request body or parameters required
    - Response: A JSON API object with the following attributes:
        i. uuid: An identifier that can be used for subsequent status requests
        ii. status: The status of the upload, will be set to initial value “pending”
        iii. location: A URL to which the actual document package payload can be submitted in the next step. The URL is specific to this upload request and should not be re-used for subsequent uploads. The URL is valid for 900 seconds (15 minutes) from the time of this response.
2. Client Request: PUT to the location URL returned in Step 2.
    - Request body should be encoded as multipart/form-data, equivalent to that generated by an HTML form submission or using “curl -F…”. The format is described in more detail below.
    - No "apikey" authorization header is required for this request, as authorization is embedded in the signed location URL.
    - Service Response: The HTTP status indicates whether the upload was successful. Additionally the response includes an ETag header containing an MD5 hash of the submitted payload. This can be compared to the submitted payload to ensure data integrity of the upload.

Multipart Format

A valid document package is a multipart/form-data element with the following parts:

Part Name	Content-Type	Contents
metadata	application/json	JSON metadata describing the individual on behalf of whom the document is being submitted, and the documents themselves. See “Metadata Fields” below.
content	application/pdf	The main document being submitted, as a valid PDF.
attachment1...attachmentN	application/pdf	(Optional) Zero or more document attachments, numbered sequentially starting with 1. Each must be a valid PDF.

Metadata Fields

Name	Type	Example	Description
veteranFirstName	String	“Jane”	Veteran first name
veteranLastName	String	“Doe”	Veteran last name
fileNumber	String	“012345678”	VA file number or SSN
zipCode	String	“97202”	Veteran zip code
source	String	“MyVSO”	(Optional) System, installation, or entity submitting the document
docType	String	“21-22”	(Optional) VBA form number of document

Example Payload

The following demonstrates a (redacted) multipart payload suitable for submitting to the PUT endpoint. Most programming languages should have provisions for assembling a multipart payload like this without having to do so manually.

```
--17de1ed8f01442b2a2d7a93506314b76
Content-Disposition: form-data; name="metadata"
Content-Type: application/json

{"veteranFirstName": "Jane",
 "veteranLastName": "Doe",
 "fileNumber": "012345678",
 "zipCode": "97202",
 "source": "MyVSO",
 "docType": "21-22"}
--17de1ed8f01442b2a2d7a93506314b76

Content-Disposition: form-data; name="content"
Content-Type: application/pdf

<Binary PDF contents>
--17de1ed8f01442b2a2d7a93506314b76

Content-Disposition: form-data; name="attachment1"
Content-Type: application/pdf
```

```
<Binary PDF attachment contents>
--17de1ed8f01442b2a2d7a93506314b76--
```

This PUT request would have an overall HTTP Content-Type header:
`"Content-Type: multipart/form-data; boundary=17de1ed8f01442b2a2d7a93506314b76"`

Note that the Content-Disposition parameter "name" in each part must be the expected values "metadata", "content", "attachment1"..."attachmentN".

Error/Retry Scenarios

The POST operation may fail with the following HTTP status codes:

- 401 Unauthorized: No API key found in request.
- 403 Forbidden: Invalid authentication credentials.
- 500 Internal Server Error

If the PUT operation fails with any non-successful HTTP status code, it *may* be retried by doing a PUT to the same location as long as the PUT location has not expired. Clients may want to do a get status operation before retrying to verify that the status still shows as "pending".

The overall upload operation can complete after the expiration time of a given location as long as the request is *initiated* before the expiration time.

It is technically possible for a client to perform multiple successful PUTs to the same location. If a client does so, only the *first* supplied payload will be processed. For this reason clients should ensure that any payloads supplied to the same location (e.g. during a retry) are semantically identical.

Status Operation

Allows a client to determine the status of an uploaded document package.

1. Client Request: GET https://api.vets.gov/services/vba_documents/v0/upload/<uuid>
2. Service Response: A JSON object with the following attributes:
    - uuid: The upload identifier as used previously.
    - status: The status of the upload (see below).
    - code: Unambiguous status code. Only present if status = "error" (see below).
    - message: Human readable error description. Only present if status = "error".
    - detail: Human readable error detail. Only present if status = "error".

In general clients will be able to check the status of an upload UUID indefinitely *if any content was uploaded for that UUID.* If a client does step 1 of the upload operation (the POST) but does

not subsequently upload any content to the provided location, then the UUID will be invalidated at some point after the 15 minute expiry of the upload location. Attempts to get status for such a UUID will result in a 404 Not Found response.

Status Values

Value	Comments
pending	Initial status, indicates no document package has been uploaded yet.
uploaded	Indicates document package has been successfully uploaded, but not yet processed or propagated to Central Mail API.
received	Indicates document package has been successfully propagated to Central Mail API.
processing	Indicates document package is being processed by ICMHS or a downstream system.
success	Indicates document package has been received by DHMS.
error	Indicates that there was an error. See the ‘code’ and ‘message’ for further information.

Note that until a document status of “received”, “processing”, or “success” is returned, a client cannot consider the document as received by VA. In particular a status of “uploaded” means that the document package has been transmitted, but possibly not validated. Any errors with the document package (unreadable PDF, etc) may cause the status to change to “error”.

Asynchronous Status Error Codes

The following codes are returned in a successful response to a "get status" operation. If the "get status" operation itself fails, see "Error Scenarios" below.

Additional error codes may be added as needed.

Code	Interpretation
DOC101	Invalid multipart payload provided - not a multipart, or missing one or more required parts
DOC102	Invalid metadata - not parseable as JSON, incorrect fields, etc
DOC103	Invalid content - not parseable as PDF. Detail field will indicate which

	content or attachment part was affected
DOC104	Submission rejected by downstream system. Detail field will indicate nature of rejection
DOC201	Upload server error
DOC202	Error during processing by downstream system. Processing failed and could not be retried. Detail field will provide additional details where available.

Error Scenarios

The get status operation itself may fail, in which case the HTTP status code should be interpreted as follows:

- 401 Unauthorized: No API key found in request.
- 403 Forbidden: Invalid authentication credentials.
- 404 Not Found: Invalid or unknown UUID.
- 500 Internal Server Error

Rationale

There are two aspects of the above API design that warrant attention:

Separate Upload Endpoint

The API requires a two step upload process: first a POST which returns a location, and then a PUT of the document payload to the indicated location. This is admittedly added complexity compared to POST'ing the payload directly to the upload endpoint.

The API gateway uses this separation to stream the document payload directly to a storage tier where it is cached for later processing. Doing so provides greater reliability and performance of the API gateway as a whole. It protects the other API endpoints from contending with the document upload process for resources: memory, network bandwidth, connections, etc.

Separating out large data payloads is a common pattern in public APIs. Examples:

- The Amazon API for uploading machine images uses a multi-step process for uploading image contents to S3 and establishing the AMI metadata.
- The Google Drive API separates out the endpoints for metadata for the upload from the endpoint used to submit the data: [https://developers.google.com/drive/v3/web/resumable-upload]
- Dropbox separates their upload endpoints to a separate hostname from the rest of their API.

- The Flickr photo upload API "*works outside the normal Flickr API framework because it involves sending binary files over the wire*": [https://www.flickr.com/services/api/upload.api.html]

Asynchronous Status

In this design a successful POST, or even a successful PUT, of the document package does not guarantee that the document was received for processing by VA. Instead clients must invoke the status endpoint for a given submission and examine the status field.

We believe this design constraint is the best representation of the inherently asynchronous nature of the document submission process as the document package passes through several hands. To try to represent that as a simple success/failure response code would be creating a leaky abstraction. Consider:

- A hypothetical synchronous submission to ICHMS might return a success status code, indicating that the document was received by ICMHS, had correct metadata, was a valid PDF, etc. But the PDF could be unreadable by a human processor - there is no automated process that interprets the actual form data along the way. So a successful submission may wind up as unprocessable.
- A hypothetical synchronous submission (which would be long-running for a large document payload) could timeout on the client side or anywhere on the network path between there and the API gateway. The client would interpret this as a failure, but the payload may have been received and been asynchronously submitted to ICMHS, processed, etc.

A synchronous API would also make it impossible for the API gateway to retry submissions to ICMHS in case that system is temporarily unavailable.

We will attempt to minimize the number of cases where a document is uploaded but cannot progress further, but representing the asynchronous nature of this submission process provides the best available information to API clients.

FAQ

Changelog

Version 0.1.0 - 2018-04-16
    Initial draft
Version 0.2.0 - 2018-04-21
    Clarified error scenarios and response formats
Version 0.3.0 - 2018-04-24

Renamed "document" part to "content"; clarified UUID retention