

**Virtual Lifetime Electronic Record (VLER)
Data Access Services (DAS)
eCRUD Service**

Version 1.0

Interface Control Document



May 2014

Revision History

Date	Version	Description	Author
10/31/2013	0.1	Initial Draft	S.Addepalli
11/04/2013	0.2	Added samples in Appendix Merged with D.Vazquez	S.Addepalli D.Vazquez
11/22/2013	0.3	Changes from peer review	F. Fontaine
11/24/2013	0.4	Added Appendix with Interface operations details	S.Addepalli
11/29/2013	0.5	Changes from peer review	F. Fontaine
04/17/2014	0.6	Updates for March, 2014	J.W. May
04/17/2014	0.7	Architect review	G.A. Ludgate
05/13/2014	0.8	Updates for May, 2014	J.W. May
05/13/2014	0.9	Architect review	G.A. Ludgate
05/15/2014	0.10	Updates from Architect Review for May, 2014	J.W. May
05/20/2014	0.11	Technical Writer Review	Nancy Burak
05/28/2014	0.12	Management Review	Cam Moore
05/28/2014	1.0	Approved	

Table of Contents

1. Introduction	1
1.1. Purpose	1
1.2. References	1
1.3. Scope	1
1.4. System Identification.....	2
1.4.1. VLER DAS.....	2
1.4.2. Interfacing Applications	2
2. Interface Definition.....	3
2.1. System Overview	3
2.2. Interface Overview	4
2.3. Data Transfer.....	5
2.3.1. Large File Upload To eCRUD	12
2.3.2. Large File Download From eCRUD.....	13
2.3.3. Custom “Create With Transform” Operations	13
2.4. Data Exchanges	14
2.4.1. HTTP “Accept” Header In Requests To eCRUD.....	14
2.4.2. HTTP “Content-Type” Header In Storage-type Requests To eCRUD	14
2.5. Performance Requirements	15
2.6. Security.....	15
Appendix A – eCRUD Request Details	16
eCRUD Operation Request URL Details	16
eCRUD Request Operations Summary	20
Appendix B – eCRUD HTTP Request Payload Examples	25
Create/POST Request STR Subscription JSON Document.....	25
Create/POST Request To “transform” Operation Request CAPRI DBQ XML Document	25
Create/POST Request To “transform” Operation Request eCFT XML Document	25
Create/POST Request STR Metadata XML Document	26
Create/POST Request STR PDF Document With Large File Upload Request ..	26
Create/POST Request “Structured audit log” JSON Document	27
Create/POST Request “Unstructured audit log” Document With Large File Upload Request.....	27
Appendix C – eCRUD HTTP Response Headers.....	28
Appendix D – eCRUD Response Body Examples	29

Read-direct/GET Single DBQ “Attachment” Document (e.g. .pdf) From GridFS Collection Response	29
Read-direct/GET Single DBQ Metadata Document From “Searchable” Collection Response	29
Read-direct/GET Single STR Subscription Document From “Searchable” Collection Response	30
Read-direct/GET Single STR Metadata Document From “Searchable” Collection Response	30
Read-direct/GET Single STR (e.g. .pdf) Document From GridFS Response.....	30
Read-direct/GET Single eCFT (e.g. .pdf) Document From GridFS Collection Response.....	31
Read-direct/GET Single eCFT Metadata Document From “Searchable” Collection Response	31
Read-direct/GET Single Document From “audit/gateway” Collection Response	31
Read-direct/GET Single Document From “audit/fs” Collection Response	32
Read-Direct/GET Single Record From “fs.files” GridFS Metadata Collection Response.....	32
Read-Query/GET Multiple Documents From “Searchable” Collection Responses.....	33
Read-Query/GET Multiple Records From “fs.files” GridFS Metadata Collection Response.....	35
Create/Update Single Document In “Searchable” Collection Success Response.....	35
Create Single Document In GridFS Collection Success Response.....	36
Delete Single Document In Searchable Collection Success Response.....	37
Delete Multiple Documents In Searchable Collection Success Response	37
Delete Single Document In GridFS Collection Success Response	37
Example Operation Failure Response (using 400/Bad Request response as an example)	38
Example Operation Failure Response (using 415/UnsupportedMediaType response as an example)	39
Example Operation Failure Response (using 404/Not Found response as an example)	39
Example Operation Error Response (using 500/Internal Server Error as an example)	40
Create/POST STR Metadata Document Success Response	40
Create/POST STR Subscription Document Success Response	41
Create/POST CAPRI And Other Partner DBQ XML Document With “transform” Operation URL Success Response	41

Create/POST eCFT XML Document With “transform” Operation URL Success Response.....	42
Appendix E – JSON to BSON Decoration Capability	43

1. Introduction

This document is the Interface Control Document (ICD) for the interface between Partners and the VLER Data Access Services (DAS) Extensible Create Read Update Delete (eCRUD) Service.

In this ICD, “partners” refers to internal and external Consumers and Producers who exchange VLER Data through the VLER DAS mechanism. External means “external to the Department of Veterans Affairs (VA) network” such as DoD while “internal” means “on the VA network”.

1.1. Purpose

This ICD serves as a specification of the interface between Partner applications and the VLER DAS eCRUD Service. It will be used by:

1. Developers of the VLER DAS eCRUD Service.
2. Developers of Partner applications.

It is assumed the reader is familiar with Reference 1 below.

1.2. References

1. [HTTP] - RFC 2616 Hypertext Transfer Protocol – HTTP/1.1, June 1999
<http://tools.ietf.org/html/rfc2616>
2. [JSON] - The JSON Data Interchange Format, ECMA-404, 1st ed., October 2013
<http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>
3. [XML] - Extensible Markup Language (XML) 1.0 (Fifth Edition), November 2008
<http://www.w3.org/TR/REC-xml/>
4. [BSON] - Binary JSON, Specification Version 1.0, retrieved April 2014
<http://bsonspec.org/spec.html>
5. [MongoDB] - “The MongoDB 2.6 Manual”, MongoDB, Inc., retrieved April 2014
<http://docs.mongodb.org/manual/>
6. [JSONPath] - “JSONPath - XPath for JSON”, by Stefan Goessner, February 21, 2007
<http://goessner.net/articles/JsonPath/>
7. [multipart/form-data] – RFC 2388 Returning Values from Forms: multipart/form-data, The Internet Society, August 1998
<http://www.ietf.org/rfc/rfc2388.txt>
8. [DCGTS] - CTS 010.001 DAS Client Gateway Technical Story - TODO
9. [DEPICD] - DAS External Partners Interface Control Document - TODO

1.3. Scope

This ICD focuses on the software interface between Partner’s applications and the VLER DAS eCRUD Service. It describes the operations (or transaction types), data transfers, and communication methods of the service interface supported by eCRUD.

Upon formal approval by each Partner application, this ICD shall be incorporated into the requirements baseline for the VLER DAS and all interfacing applications.

1.4. System Identification

1.4.1. VLER DAS

System	Details
Identification number	VLER DAS
Title	Virtual Lifetime Electronic Record Data Access Service
Abbreviation	VLER DAS
Version number	1
Release number	4.4.4
Point of Contact	John W. May

1.4.2. Interfacing Applications

Application Name	Interface Definition
VBMS	VLER-DAS – Subscription Service, VLER-DAS -- eCRUD
IDES/eCFT	VLER-DAS -- eCRUD
QTC	VLER-DAS -- eCRUD
LHI	VLER-DAS -- eCRUD
Medical Services of Los Angeles (MSLA)	VLER-DAS -- eCRUD
VES	VLER-DAS -- eCRUD
DAS Client Gateway	VLER-DAS -- eCRUD
VLER-DAS – BTS Service	VLER-DAS -- eCRUD
VLER-DAS – Subscription Service	VLER-DAS -- eCRUD
VLER-DAS – Pre-fetch Service	VLER-DAS -- eCRUD
VLER-DAS – HTTP Scheduler	VLER-DAS -- eCRUD
REDIS	VLER-DAS -- eCRUD
MongoDB	VLER-DAS -- eCRUD

2. Interface Definition

VLER DAS provides the eCRUD Service which can be used by partners to store, update, delete, or get documents in or from the VLER Data Store.

This ICD describes the “Version 1” interface to eCRUD. This version number is an essential part of the interface URL as shown in Appendix A.

2.1. System Overview

The VLER DAS serves as a broker between Partners which can be Consumers or Producers of VLER data. Partners initiate all data transactions for this interface. The diagram (Figure 1- DAS Application Architecture) shows a data flow diagram of data exchanges that happen between the various Consumers and Producers through the set of VLER DAS applications. The bubble labeled “Persistence as a Service Enterprise CRUD” represents the eCRUD Service, and the box labeled “VLER Data Store” (VDS) represents an instance of a MongoDB database.

VLER DAS performs this main business function:

- Bidirectional exchange and storage of structured, semi-structured, and unstructured information both within the VA and, in general, between all VLER Partners.

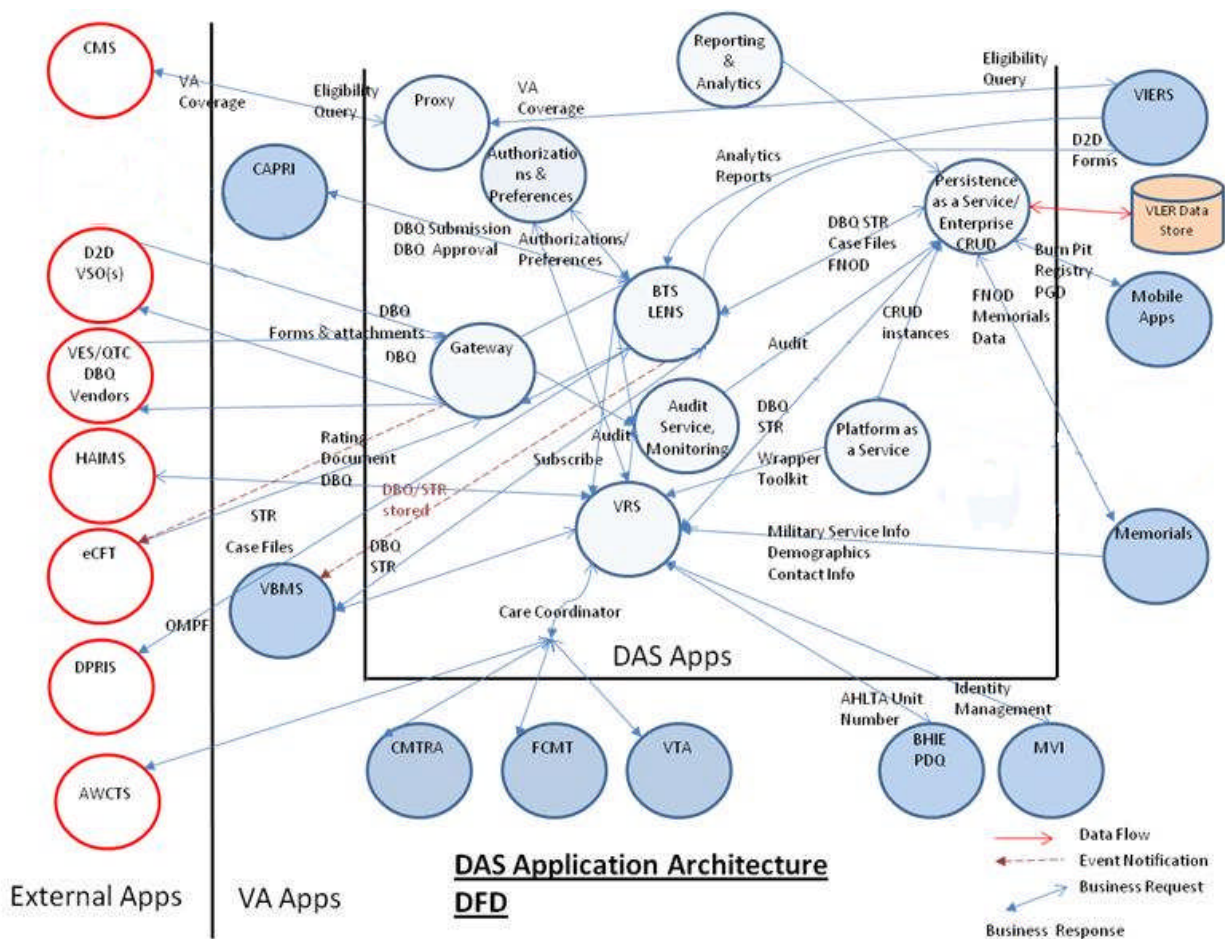


Figure 2- DAS Application Architecture

2.2. Interface Overview

Enterprise eCRUD provides RESTful APIs to Partners to perform create, read, update or delete (CRUD) operations on VLER DAS data stored in the VLER Data Store . It also supports numerous adapters for data transformation, notification of data changes and custom event handlers. All access to data is checked for authorization and audited by the Client Gateway for external Partners (see also the External Partners ICD when available).

eCRUD will allow Partners to store binary data for a specific, well-known MIME data types when possible, or else stored as arbitrary binary data with the “application/octet-stream” MIME data type by default. eCRUD supports XML and JSON. It also supports real-time lookup and the simple querying of structured data (Key/Value Pairs).

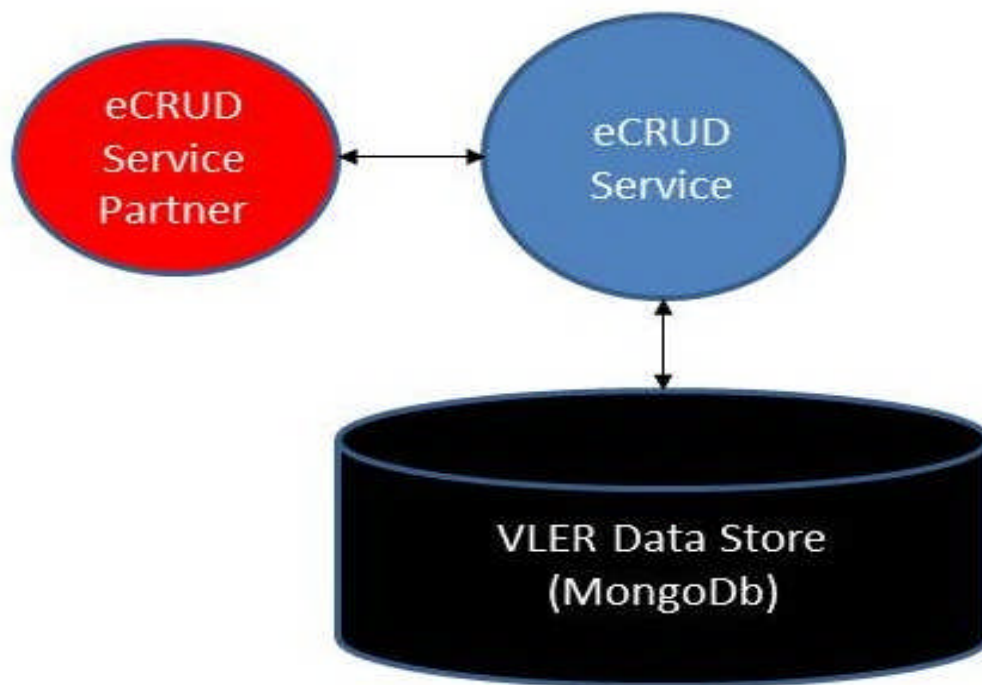


Figure 3 eCRUD Service Data Flow Diagram



eCRUDICDDiag.pptx

eCRUD Service Implementation

The eCRUD Service is implemented in JavaScript and executes under Node.js. It provides a RESTful web service interface for a Partner to perform CRUD database operations on the VLER Data Store. The VLER Data Store is implemented with a MongoDB database. Files smaller than 16MB, often searchable text data, are stored in “searchable” collections in MongoDB, and stored in the JSON format. Also, within MongoDB, a binary file partition known as the “GridFS partition”, or merely as “GridFS”, is used to store binary and large-size files, i.e. over 16MB in size. These large-size files are streamed to and from the database in binary streams so that they do not have to occupy large amounts of memory at any given time in eCRUD.

2.3. Data Transfer

The eCRUD request URLs are formatted according to the convention described in Appendix A, section 3.1.

General eCRUD Service HTTP Request Description

The basic or “generic” type of supported eCRUD operations for smaller documents (under 16MB in size), usually “structured” searchable text, are shown below in a table, with

1. specific HTTP request methods,
2. URL formats,
3. relevant HTTP Request Headers

and other information. Note that the “collectionName” in these operations never have the “fs” value in the URL. These type of collections are known in the rest of the document as “searchable” collections.

where:

{host} is the IP address or DNS hostname of the application server on which eCRUD Service is executing,

{port} is the server port number of the eCRUD Service running on the given “host”, and
<dbname> is the MongoDB database name representing the database instance on which the “collectionName” exists (and currently must either be “core” or “audit”).

Table 1 -- General eCRUD "Searchable" Collection Request Operations

eCRUD Operation	HTTP Request Method	Example URL Format	Required Request Header Parameters
Create	POST	<a href="http(s)://{host}:{port}/ecrud/v1/<dbName>/<collectionName>">http(s):// {host}:{port}/ecrud/v1/<dbName>/<collectionName> ≥	Content-Type: application/json,, application/xml, plain/text, application/x- www-form- urlencoded; Accept: application/json, application/xml
Custom Create With Transform (for embedded base-64 attachment separation to GridFS. Etc.)	POST	http(s):// {host}:{port}/ecrud/v1/core/disabilityBenefitsQuesti onnaire/transform or: http(s):// {host}:{port}/ecrud/v1/core/electronicCaseFiles/tran sform	Content-Type= multipart/form-data (NOTE: The attachment name must also contain “file”). See Reference 7 on multipart/form-data for more information. Accept: application/json, application/xml

eCRUD Operation	HTTP Request Method	Example URL Format	Required Request Header Parameters
Read - direct	GET	<a href="http(s)://{host}:{port}/ecrud/v1/<dbName>/<collectionName>/<fileId>[?jpath={jpathQueryString}]">http(s)://{host}:{port}/ecrud/v1/<dbName>/<collectionName>/<fileId>[?jpath={jpathQueryString}]	Accept: application/json, application/xml (Accept: application/json is default)
Read - query	GET	<a href="http(s)://{host}:{port}/ecrud/v1/<dbName>/<collectionName>[?query={queryString}][&limit={maxRecords}&fields={columnProjections}&sort={sortCriteria}]">http(s)://{host}:{port}/ecrud/v1/<dbName>/<collectionName>[?query={queryString}][&limit={maxRecords}&fields={columnProjections}&sort={sortCriteria}]	Accept: application/json, application/xml (Accept: application/json is default)
Update	PUT	<a href="http(s)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>/<fileId>[?upsert={booleanValue}&multi={booleanValue}&query={queryString}]">http(s)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>/<fileId>[?upsert={booleanValue}&multi={booleanValue}&query={queryString}]	Content-Type: application/json, application/xml, plain/text, application/x- www-form- urlencoded; Accept: application/json, application/xml
Remove/Delete Document	DELETE	<a href="http(s)://{host}:{port}/ecrud/v1/<dbName>/<collectionName>/<fileId>">http(s)://{host}:{port}/ecrud/v1/<dbName>/<collectionName>/<fileId>	Accept: application/json, application/xml
Remove Multiple/Bulk Delete	DELETE	<a href="http(s)://{host}:{port}/ecrud/v1/<dbName>/<collectionName>?query={queryString}">http(s)://{host}:{port}/ecrud/v1/<dbName>/<collectionName>?query={queryString}	Accept: application/json, application/xml

Files to be stored using the above URL/operations must be included in the Request body of an HTTP POST Request to the appropriate eCRUD URL. These files are stored in the GridFS partition of the database, where they are stored as binary “chunks” in the one “<dbName>/fs” collection.

Requests to store data directly into the GridFS binary large-file storage partition requires the use of distinct URL formats and optional HTTP Request Header combinations. The general form of these are listed in the following table:

Table 2 -- General eCRUD Large File (GridFS) Request Operations

eCRUD Operation	HTTP Request Type	Example URL Format	Required Request Header Parameters
Create (i.e. Large File Upload)	POST	<a href="http(s)://{host}:{port}/ecrud/v1/<dbName>/fs">http(s)://{host}:{port}/ecrud/v1/<dbName>/fs NOTE: There is but one collection for binary data	Content-Type=multipart/form-data (NOTE: The attachment name must also contain “file”). See Reference 7 on multipart/form-data for more information. Accept: application/json, application/xml
Read – direct (i.e. Large File Download)	GET	<a href="http(s)://{host}:{port}/ecrud/v1/<dbName>/fs/<fileId>≥">http(s)://{host}:{port}/ecrud/v1/<dbName>/fs/<fileId>≥	N/A
Read - query	GET	<a href="http(s)://{host}:{port}/ecrud/v1/<dbName>/fs.files[?query={queryString}][&limit={maxRecords}&fields={columnProjections}&sort={sortCriteria}]">http(s)://{host}:{port}/ecrud/v1/<dbName>/fs.files[?query={queryString}][&limit={maxRecords}&fields={columnProjections}&sort={sortCriteria}]	Accept: application/json, application/xml (Accept: application/json is default)
Update	PUT	<a href="http(s)://{host}:{port}/ecrud/v1/<dbName>/fs/<fileId>≥[?upsert={booleanValue}]">http(s)://{host}:{port}/ecrud/v1/<dbName>/fs/<fileId>≥[?upsert={booleanValue}]	Content-Type=multipart/form-data (NOTE: The attachment name must also contain “file”). See Reference 7 on multipart/form-data for more information. Accept: application/json, application/xml
Remove/Delete	DELETE	<a href="http(s)://{host}:{port}/ecrud/v1/<dbName>/fs/<fileId>≥">http(s)://{host}:{port}/ecrud/v1/<dbName>/fs/<fileId>≥	Accept: application/json, application/xml

The “Content-Type” header is included in the HTTP Request Headers to specify the MIME-type of the contents in the HTTP Request Body being stored or updated in the database. For the eCRUD Create and Update operations to “searchable” collections, for example, the “Content-Type” header value can be “application/xml”, “text/xml”, “application/json”, or “text/plain”.

For GridFS Create and Update operations, the “Content-Type” header must always be “multipart/form-data”. The attachment name in this request must contain the text “file” somewhere in it, e.g. “filename”. See Reference 7 on sending MIME multipart/form-data HTTP requests.

An “Accept” header is included in the HTTP Request to specify the desired MIME-type of the HTTP Response Body returned to the eCRUD Service Partner. For the Create and Read, Update and Delete operations, for example, this “Accept” header value can be “application/xml”, or “application/json”, where “application/json” is the default value for the response returned if no “Accept” header is set. For the GridFS Read operation request, the “Accept” header is left unset. See section 2.4.1 below for more information.

The Update/PUT operation for a “searchable” collection document can be performed in one of two basic ways: 1) either by specify the document to update with a <fileId>, or 2) by select one or more documents to update with the “query=” (and if multiple documents are to be updated, with the “multi=true” parameter included in the request URL).

The Update/PUT operation for a GridFS collection document can only be done using a <fileId> in the request URL.

If the “upsert=” parameter is set to “true”, then the Update/PUT operation will create a new document with a Create operation when the <fileId> or “query=” selection criteria do not find a match in the collection. The default “upsert=” value is “false”.

See Appendix A, section 3.1 and section 3.2 for more on the entire document vs. specific-fields-only Update operations for “searchable” collections.

The Delete Request for both MongoDB and the GridFS partition use the “Accept” header to specify the MIME type of the HTTP response. The value can be either application/json or application/xml. Application/json is assumed if the Accept header is not provided.

See Appendix A for more specific details of all eCRUD Service request operations.

Special “Create” Operation Considerations for MongoDB and GridFS

eCRUD has the facility to allow a new collection to be created by a single Create operation. The collection name will be set to the collection name used in the POST URL. This means that any eCRUD Partner can create a brand new “databaseName/collectionName”-named collection with eCRUD. Care should be taken to avoid mistakes in the URL “collectionName” parameter, so that the storage in eCRUD occurs as expected. Even a one character difference will create a whole new eCRUD collection as will the use of different upper or lower case letters in the collection name.

All records stored in MongoDB are assigned a unique ID. This ID is stored in an element called “_id”. If an “_id” field is included at the root level in the document to be stored in a Create operation on a “searchable” collection, then MongoDB will use this value, if possible, as the unique identifier. (If no “_id” field is present at the root level, then MongoDB will automatically generate a 24-character hex “_id” value instead).

eCRUD Response HTTP Response Status Values

The eCRUD Service returns an HTTP response for every request. The response contains one of the following HTTP status codes.

Table 3 -- eCRUD Service HTTP Response Status Values

Response Type	HTTP Status Code / Reason Phrase
Success	200 OK (Read and Delete operations) 201 Created (Create and Update operations)
Success	204 No Content - response to a query that returns no content
Failure	400 Bad Request – when, say, syntax of query is bad or a query is specified on an operation for which it makes no sense.
Failure	404 Not Found – the requested resource was not located
Failure	415 Unsupported Media Type – when the Content-Type header of a Create or Update operation request has an unsupported media type value
Error	500 Internal Server Error – the default internal error for an unspecified reason
Error	503 Service Unavailable – for when MongoDB is down or network to mongo is down etc
Error	504 Gateway Time-out - if the MongoDB client request time-out expires, or another internal time-out expires

Note: A response of “502/Bad Gateway” may also be received by external Partners when accessing eCRUD. This is because External Partners must access eCRUD through the DAS Client Gateway proxy. DAS Client Gateway may return the “502/Bad Gateway” response to an external Partner for an eCRUD request that is not responded to by eCRUD within a configured timeout duration (see also the External Partners ICD when available).

General eCRUD Service HTTP Response Description

The desired response MIME-type for certain eCRUD operations is specified in the HTTP Request “Accept” header (see table above). The supported eCRUD response MIME-types, and the description of the response returned, for the relevant operations which return a response, are shown in the following table:

Table 4 -- General eCRUD Response Descriptions By Operation Type

eCRUD Operation	HTTP Request Accept Header Value(s)	Success Response Body Description
Create	application/xml, application/json. No “Accept” header returns application/json by default.	Depending on the MIME type specified in Accept header, the response will contain a JSON or XML structure that signals the success of the create.
Create with transform	Same as “Create”	Same as “Create”
Read - direct	application/xml, application/json, No “Accept” header returns application/json by default	Document stored in MongoDB, and named by documentID returned in the MIME-type format specified in the Accept HTTP Request Header (e.g. in XML, or in JSON), XML is enclosed in by <document></document>
Read - query	application/xml, application/json. No “Accept” header returns application/json by default	List of documents stored in MongoDB, matching the query string if any, which has been used as the query parameter in MongoDB find() call, stored in MongoDB, returned in the MIME-type format specified in the Accept HTTP Request Header parameter (e.g. in XML, or in JSON), When XML is returned, each result is enclosed by <document></document> and the entire resultset is enclosed by <documents></documents>
Update	Same as “Create”	Same as “Create”
Remove	application/xml, application/json. No “Accept” header returns application/json by default	Returns the JSON object {“success” : “true”}, (or in xml format when specified)

For all responses returned by the eCRUD Service, the Service has the capability to convert the response body from the stored, JSON format into XML. eCRUD receives the instruction to do this by setting the “Accept” HTTP Request Header to the “application/xml” value (see above). This is useful because MongoDB stores all documents only in the JSON format. The default response format-type is JSON if none is specified in the “Accept” request header.

In the response to a successful Read-direct operation, eCRUD will return the single document – either from a “searchable” collection or from GridFS – which matches the <fileId> in the request URL.

In the response to a successful Read-query operation, eCRUD will return all documents which match the query condition(s) specified, and if a “limit=” query parameter is not set in the request URL, up to the default, configured document limit quantity, currently configured to a default limit of 100 documents. The default limit to the quantity of result documents can be overridden to be greater than the default (i.e. 100) by explicitly setting a “limit=” parameter in the request URL.

In the response to a successful Create/POST or Update/PUT request, eCRUD will return the unique “_id” value which names the file in VDS/MongoDB, as well as an “uploadDate” value which represents the date and time the document was created or updated in the VLER Data Store (in GMT). See the examples in Appendix D, sections 6.13 and 6.14.

For a successful DELETE request, eCRUD will return the “success” response in Appendix D, section 6.15.

See also the rest of Appendix D for more eCRUD Service response examples.

See also Appendix C for a description of the HTTP Response Headers returned from the eCRUD Service.

eCRUD Service Environment External Base URLs

The following base URLs are to be used for DAS-external hosts to access different VLER DAS environments. SSL Certificates must be exchanged with VLER DAS operations personnel before testing in the Integration Testing environment (i.e. “SILVER” environment), testing in the SQA environment (i.e. “GOLD” environment), or production service commences (see also the External Partners ICD when available).

Table 5 -- DAS Environment Protocols And External Hostnames

Environment	Environment Name	URL Protocols and External Hosts
Integration Testing	SILVER	https://SILVERVLER.VA.GOV
SQA/EndToEnd testing	GOLD	https://GOLDVLER.VA.GOV
Production	PRODUCTION	https://VLER.VA.GOV

2.3.1. Large File Upload To eCRUD

Large files for eCRUD are defined as larger than 16MB, as MongoDB handles files of this size or greater differently by requiring them to be stored in the “GridFS Partition” collection, i.e. “GridFS”, rather than a “searchable” collection. The currently supported, maximum large file size which can be uploaded to eCRUD is 500 MB.

To upload a large file into GridFS:

1. ensure that the collection name is "fs",
2. the HTTP “Content-Type” header is set to “multipart/form-data”, and
3. the MIME “multipart/form-data” request body must be sent as the POST request body, including ensuring that the name field of the file being uploaded is “file” (See Reference 7 on MIME multipart/form-data requests for more information). Note that eCRUD currently only accepts one “part” in a “multipart”-type POST request, and this “part” must be the first “part” in the POST request body.

An example HTTP request is:

POST <https://{host}:{port}/ecrud/v1/<dbName>/fs> HTTP/1.1

It is recommended but not required that the “Content-Length” HTTP request header value is set for this type of “large file upload” request to eCRUD.

2.3.2. Large File Download From eCRUD

Large files for eCRUD are defined as those larger than 16MB. MongoDB handles files of this size or greater differently by requiring them to be stored in the GridFS Partition, i.e. “GridFS”, rather than in a “searchable” collection.

To get the list of available files from GridFSs use the following HTTP(S) request (see Appendix D, section 6.10 Read-Direct/GET Single Record From “fs.files” GridFS Metadata Collection Response for more information and an example):

GET [http\(s\)://{host}:{port}/ecrud/v1/<dbName>/fs.files](http(s)://{host}:{port}/ecrud/v1/<dbName>/fs.files)

For downloading a single file, use the following HTTP(S) request:

GET [http\(s\)://{host}:{port}/ecrud/v1/fs/<fileId>](http(s)://{host}:{port}/ecrud/v1/fs/<fileId>)

For these single file download requests, there are three notable headers included with the standard eCRUD response headers: Content-Length, Content-Type, and Content-Disposition. Content-Length is the length of the binary file being downloaded, Content-Type is the MIME-type of the file being downloaded, and Content-Disposition contains the original filename of the file being downloaded (See Appendix C for more information).

The response body for a single file download request is returned as binary to the Consumer.

2.3.3. Custom “Create With Transform” Operations

The eCRUD Service provides the capability to create documents in both the “searchable” collections and the GridFS collections with a single Create/POST request. The request is made to a URL with a “transform” modifier at the end of the URL Path, having the form:

<http://{host}:{port}/ecrud/v1/{databaseName}/{searchableCollectionName}/transform>

Currently there are two “Custom Create With Transform” operations supported in eCRUD: one for Creating DBQ documents, using “disabilityBenefitsQuestionnaires” as the value for the “searchableCollectionName”, and one for Creating eCFT documents, using “electronicCaseFiles” as the value for the “searchableCollectionName”. This single request contains either the DBQ or eCFT XML document, and this XML document as an “attachment” element within it of a specified type, e.g. application/pdf, yet with the data of the attachment stored in the base 64 encoding format. eCRUD will receive the Create/POST request containing this XML document with its base 64-encoded attachment, removes and decodes the attachment from base 64-encoding to a binary file, stores this now-binary attachment file into GridFS, retrieves the automatically created “_id” in the response from MongoDB which names this attachment file and places this into the XML document where the base 64-encoding originally was located, and then will store this finalized XML document into the specified “searchableCollectionName” collection specified in the request URL. This XML document in the “searchableCollectionName” collection becomes a searchable “metadata” for the binary attachment file stored in GridFS. In the case of DBQ documents stored with this Custom Transform method, a copy of the original DBQ XML document with the original base64-encoded attachment is also stored.

For examples of requests to the “transform” URL operations, see Appendix B, sections “4.2 - Create/POST Request To “transform” Operation Request CAPRI DBQ XML Document” and “4.3 - Create/POST Request To “transform” Operation Request eCFT DBQ XML Document”. For examples of content available in “searchable” collections from the “transform” URL operations, see Appendix D, sections “6.24 - Create/POST CAPRI DBQ XML Document With “transform” Operation

URL Success Response", and "6.25 - Create/POST eCFT DBQ XML Document With “transform” Operation URL Success Response".

Currently the “Content-Desc” HTTP request header is an eCRUD-specific request header used to specify to the eCRUD Service that the XML document in the “Custom Create With Transform” operations contains a certain XML format. Currently the “niem/xml” is the only value used to specify “NIEM”-schema XML documents, but more values may be added in future releases. The “niem/xml” value is the default value for all “Custom Create With Transform” operations.

Note that the Update/PUT operation cannot be used with the “transform” operation URL.

2.4. Data Exchanges

2.4.1. HTTP “Accept” Header In Requests To eCRUD

The HTTP “Accept” header can be used to tailor the type of response returned by a request to the eCRUD Service.

By default, the eCRUD Service will return a response in the JSON-format, and with a response “Content-Type” header value of “application/json”. If the consumer specifies an HTTP “Accept” header in the request to eCRUD with a value of “application/xml”, then all the responses are converted from JSON to XML, and are returned with a response “Content-Type” header value of “application/xml”.

2.4.2. HTTP “Content-Type” Header In Storage-type Requests To eCRUD

“Smaller-size” documents (i.e. under 16MB in size) posted to “searchable”, non-GridFS collections (i.e. those collections without “fs” as the collection name) in either POST or PUT requests to eCRUD are always stored in BSON-document side of MongoDB in the “MongoDB”-specific flavor of the JSON format called BSON (Reference 4). The differences between BSON and JSON are minor and are mainly to accommodate MongoDB database-specific data types (see Appendix E). eCRUD can convert request-body data from the XML, form-encoded, or text formats into the JSON-like BSON format so the data can be stored in the BSON-document side of MongoDB (since a non-GridFS collection document is always stored in the JSON/BSON format in MongoDB). The type of data in the request body for storage is specified using the HTTP “Content-Type” header in the request to the eCRUD Service. The following table outlines the values that can be passed in the request’s “Content-Type” header.

See section 2.4.1 and Reference 7 for more on using the “multipart/form-data” to upload large files (i.e. equal to or greater than 16MB in size) to the GridFS collection via eCRUD with POST and PUT requests. The following table summarizes all of the supported types of Content-Type values for eCRUD for storage-type requests, which include the POST and PUT requests:

Table 6 -- Supported “Content-Type” Request Header Values For eCRUD Create and Update Operations

Content- Type Header Value in Request To eCRUD	Description
application/xml	Input is an XML document.
application/json	Input is an JSON document. (default)
application/x-www-form-urlencoded	Input is a set of key/value pairs that have been URL encoded. This format is handy if the input was provided via an HTML form.
text/plain	Input is a block of text.
multipart/form-data	This input format is used to post large files to the fs collection (see section 2.3.1 above), or to a non-GridFS collection when using the “transform” feature. See Reference 7 for more information on MIME multipart/form-data requests.

2.5. Performance Requirements

The following table outlines the performance-related requirements to which the eCRUD Service must conform:

Table 7 -- eCRUD Service Performance Requirements

Protocol Parameter	Value
Average Load (transactions per second)	TBD
Peak Load (transactions per second)	TBD
Response time in Second	TBD
Request Timeout Period	TBD

2.6. Security

DAS-external transfers to and from eCRUD are protected by HTTPS using certificates which identify both Partner and VLER DAS computers. The certificates for the external application and the VLER DAS computers must be securely exchanged with VLER DAS Operations team before the environments can be used.

Appendix A – eCRUD Request Details

eCRUD Operation Request URL Details

This appendix is included to provide a set of request operations to help describe the eCRUD Service interface and operations in detail.

1. The RESTful request URL format for the eCRUD Read-query operations (i.e. GET search requests) is:

`http(s)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>[?query={queryString)][&limit={maxRecords}&fields={columnProjections}&sort={sortCriteria}]`

where:

- eCRUD Service can be exposed using either HTTP for DAS-local applications, and the HTTPS protocol for DAS-external applications (those which must use the DAS-external base URLs defined in Table 5).
- {host} - domain name/IP address of the server hosting the application.
- {port} - port number listening to the incoming messages.
- <version> - interface version number.
- <dbName> - the VLER Data Store (VDS) MongoDB database name where the data for this operation is to be read.
- <collectionName> - for MongoDB read requests: this is the VDS, MongoDB collection name, for GridFS partition, this will always have the value of “fs”.
- [] indicates an optional field
- ?query={queryString} - the optional “query=” request parameter, accepts a {queryString} which is a key-value pair of the selection criteria in a JSON format.
- &limit={maxRecords} - the optional “limit=” request integer parameter, accepts a {maxRecords} value which sets the maximum number of results that can be sent back in the response. When not specified: will default to 100 max records, when zero: implies no limits and all records will be returned and when negative: the absolute value will be considered.
- &fields={columnProjections} - the optional “fields=” request parameter, accepts a {columnProjections} value which sets the fields that can be included/not included in the response. When not specified will return all fields in the document.
- &sort={sortCriteria} - the optional “sort=” request parameter, accepts a {sortCriteria} value which sets the sort criteria on field(s) in ascending or descending order. When not specified will be defaulted to the ascending order of the collection.

For more details on the “query=” parameter see the MongoDB Manual website (Reference 5) for the equivalent MongoDB “find()” command and it’s “criteria” or “query criteria” parameter at:

<http://docs.mongodb.org/manual/reference/method/db.collection.find/#db.collection.find>

For more details on the “limit=” parameter see the MongoDB Manual website (Reference 5) for the equivalent MongoDB cursor.limit() command at:

<http://docs.mongodb.org/manual/reference/method/cursor.limit/#cursor.limit>

For more details on the “fields=” parameter see the MongoDB Manual website (Reference 5) for the equivalent MongoDB “find()” command and it’s “projection” parameter details under the “Projections” header at:

<http://docs.mongodb.org/manual/core/read-operations-introduction/#projections>

For more details on the “sort=” parameter see the MongoDB Manual website (Reference 5) for the equivalent MongoDB “cursor.sort()” command at:

<http://docs.mongodb.org/manual/reference/method/cursor.sort/#cursor.sort>

2. The RESTful request URL format for the eCRUD Read-direct operation (i.e. GET request to download a document) is:

**[http\(s\)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>/<fileId>\[?
jpath={jpathQueryString}&fields={columnProjections}\]](http(s)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>/<fileId>[?jpath={jpathQueryString}&fields={columnProjections}])**

where:

- eCRUD Service can be exposed using either HTTP for DAS-local applications, and the HTTPS protocol for DAS-external applications (those which must use the DAS-external base URLs defined in Table 5).
- {host} - domain name/IP address of the server hosting the application.
- {port} - port number listening to the incoming messages.
- <version> - interface version number.
- <dbName> - the VLER Data Store (VDS) MongoDB database name where the data for this operation is to be read.
- <collectionName> - for MongoDB “searchable” collection requests: this is the VDS, MongoDB collection name, for GridFS partition, this will always have the value of “fs”.
- [] indicates an optional field.
- <fileId> - the optional VDS MongoDB or GridFS file id within the specified collection.
- ?jpath={jpathQueryString} - the optional “jpath=” request parameter, accepts a {jpathQueryString} which is a JSONPath query string value, which is JSON-type ofXPath-like request (e.g. %22\$..author%22) – This is a way to filter the result JSON response and return only the part of a “searchable” collection JSON document referenced by <fileId> which matches this query condition. This is not normally used with the fields= query parameter.
- &fields={columnProjections} - the optional “fields=” request parameter, accepts a {columnProjections} value which sets the fields that can be included/not included in the response. When not specified will return all fields in the document. This is not normally used with the jpath= query parameter.

For more details on the “fields=” parameter see the MongoDB Manual website (Reference 5) for the equivalent MongoDB “find()” command and it’s “projection” parameter details under the “Projections” header at:

<http://docs.mongodb.org/manual/core/read-operations-introduction/#projections>

For more details on the “jpath=” parameter <jpathQueryString> value possibilities see the JSONPath information website by Stefan Goessner at:

<http://goessner.net/articles/JsonPath/>

3. The RESTful request URL format for the eCRUD “Searchable” collection Create operations (i.e. POST request) for files under 16MB in size is:

[http\(s\)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>](http(s)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>)

or for GridFS-direct Create operations (i.e. POST request) recommended for files over 16MB in size is:

[http\(s\)://{host}:{port}/ecrud/<version>/<dbName>/fs](http(s)://{host}:{port}/ecrud/<version>/<dbName>/fs)

where:

- eCRUD Service can be exposed using either HTTP for DAS-local applications, and the HTTPS protocol for DAS-external applications (those which must use the DAS-external base URLs defined in Table 5).
- {host} - domain name/IP address of the server hosting the application.
- {port} - port number listening to the incoming messages.
- <version> - interface version number.
- <dbName> - the VLER Data Store (VDS) MongoDB database name where the data for this operation is to be read.
- <collectionName> - for MongoDB “searchable” collection requests: this is the VDS, MongoDB collection name, for GridFS partition, this will always have the value of “fs”.

4. The RESTful request URL format for the eCRUD Single Document Delete operation (i.e. DELETE request) is:

[http\(s\)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>/<fileId>](http(s)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>/<fileId>)

where:

- eCRUD Service can be exposed using either HTTP for DAS-local applications, and the HTTPS protocol for DAS-external applications.
- {host} - domain name/IP address of the server hosting the application.
- {port} - port number listening to the incoming messages.
- <version> - interface version number.
- <dbName> - the VLER Data Store (VDS) MongoDB database name where the data for this operation is to be read.
- <collectionName> - for MongoDB “searchable” collection requests: this is the VDS, MongoDB collection name, for GridFS partition, this will always have the value of “fs”.
- <fileId> - the optional VDS MongoDB or GridFS file id within the specified collection.

5. The RESTful request URL format for the eCRUD Multiple Document Delete operation (i.e. DELETE request) is:

[http\(s\)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>?query={queryString}](http(s)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>?query={queryString})

where:

- eCRUD Service can be exposed using either HTTP for DAS-local applications, and the HTTPS protocol for DAS-external applications (those which must use the DAS-external base URLs defined in Table 5).
- {host} - domain name/IP address of the server hosting the application.
- {port} - port number listening to the incoming messages.
- <version> - interface version number.
- <dbName> - the VLER Data Store (VDS) MongoDB database name where the data for this operation is to be read.
- <collectionName> - for MongoDB “searchable” collection requests: this is the VDS, MongoDB collection name. Note that for GridFS partition, this operation is not supported.
- ?query={queryString} - the optional “query=” request parameter, accepts a {queryString} which is a key-value pair of the selection criteria in a JSON format.

6. The RESTful request URL format for the eCRUD Single Document Update operations (i.e. PUT requests) for “searchable” collections is:

[http\(s\)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>/<fileId>](http(s)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>/<fileId>)

where:

- eCRUD Service can be exposed using either HTTP for DAS-local applications, and the HTTPS protocol for DAS-external applications (those which must use the DAS-external base URLs defined in Table 5).
- {host} - domain name/IP address of the server hosting the application.
- {port} - port number listening to the incoming messages.
- <version> - interface version number.
- <dbName> - the VLER Data Store (VDS) MongoDB database name where the data for this operation is to be read.
- <collectionName> - for MongoDB “searchable” collection requests: this is the VDS, MongoDB collection name, for GridFS partition, this will always have the value of “fs”.
- <fileId> - the VDS MongoDB file _id within the specified “searchable” collection, which is used to specify a document to update in the Update operation. Note that for GridFS partition, this operation is not supported.

7. The RESTful request URL format for the eCRUD Custom “transform” Create operations (i.e. POST request) is:

[http\(s\)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>/transform](http(s)://{host}:{port}/ecrud/<version>/<dbName>/<collectionName>/transform)

where:

- All is the same as the eCRUD Create “Searchable” collection Create operation above (item 3.), except:

- transform – signifies to eCRUD that this is a request to the Custom “transform” adapter linked to the given <collectionName> in the same request URL.

eCRUD Request Operations Summary

See Table 1 below for a summary of eCRUD Service request operations using the request URL format in section 3.1 above:

Table 8 -- eCRUD Request Operations Summary

Operation Name	HTTP Method	URL Path and Query Parameters	HTTP Request Headers, Special Requirements
Electronic Case Files storage, with Custom “transform” adapter	POST	/ecrud/v1/core/electronicCaseFiles/transform	Content-Type=multipart/form-data (NOTE: The attachment name must also contain “file”). See Reference 7 on multipart/form-data for more information. Accept="application/json" or "application/xml" (default is "application/json")
Disability Benefits Questionnaires storage, with Custom “transform” adapter	POST	/ecrud/v1/core/disabilityBenefitsQuestionnaires/transform	Content-Type=multipart/form-data (NOTE: The attachment name must also contain “file”). See Reference 7 on multipart/form-data for more information. Accept="application/json" or "application/xml" (default is "application/json")
Generic storage with Custom “transform” adapter	POST	<a href="/ecrud/v1/<dbName>/<collectionName>/transform">/ecrud/v1/<dbName>/<collectionName>/transform	Content-Type=multipart/form-data (NOTE: The attachment name must also contain “file”). See Reference 7 on multipart/form-data for more information. Accept="application/json" or "application/xml" (default is "application/json")

STR/LENS Subscriptions storage	POST	/ecrud/v1/core/serviceTreatmentRecords.subscriptions	Content-Type="application/xml" Accept="application/json" or "application/xml" (default is "application/json")
STR metadata storage	POST	/ecrud/v1/core/serviceTreatmentRecords	Content-Type="application/xml" Accept="application/json" or "application/xml" (default is "application/json")
Core Direct-To-GridFS document storage, large file upload	POST	/ecrud/v1/core/fs	Content-Type=multipart/form-data (NOTE: The attachment name must also contain "file"). See Reference 7 on multipart/form-data for more information. Accept="application/json" or "application/xml" (default is "application/json")
Audit Log "Structured" Entry from Client Gateway document storage	POST	/ecrud/v1/audit/gateway	Content-Type="application/json" Accept="application/json" or "application/xml" (default is "application/json")
Audit Log "Unstructured" Entry document storage (to GridFS)	POST	/ecrud/v1/audit/fs	Content-Type=multipart/form-data (NOTE: The attachment name must also contain "file"). See Reference 7 on multipart/form-data for more information. Accept="application/json" or "application/xml" (default is "application/json")

Generic "Searchable" Collection document storage	POST	/ecrud/v1/<dbName>/<collectionName>	Content-Type="application/json" or "application/xml" or "plain/text" or "application/x-www-form-urlencoded" Accept="application/json" or "application/xml" (default is "application/json")
Core Direct-To-GridFS document storage, large file upload	POST	/ecrud/v1/<dbName>/fs	Content-Type=multipart/form-data (NOTE: The attachment name must also contain "file"). See Reference 7 on multipart/form-data for more information. Accept="application/json" or "application/xml" (default is "application/json")
Electronic Case Files collection document retrieval	GET	/ecrud/v1/core/electronicCaseFiles/<fileId>[?jpath={jpathQueryString}]	Accept="application/json" or "application/xml" (default is "application/json")
Disability Benefits Questionnaires collection document retrieval	GET	/ecrud/v1/core/disabilityBenefitsQuestionnaires/<fileId>[?jpath={jpathQueryString}]	Accept="application/json" or "application/xml" (default is "application/json")
STR/LENS Subscriptions collection document retrieval	GET	/ecrud/v1/core/serviceTreatmentRecords.subscriptions/<fileId>[?jpath={jpathQueryString}]	Accept="application/json" or "application/xml" (default is "application/json")
STR metadata collection document retrieval	GET	/ecrud/v1/core/serviceTreatmentRecords/<fileId>[?jpath={jpathQueryString}]	Accept="application/json" or "application/xml" (default is "application/json")
Core Direct-To-GridFS document retrieval, large file download	GET	/ecrud/v1/core/fs/<fileId>	Accept header is ignored
Audit Log "Structured" Entry from Client Gateway collection document retrieval	GET	/ecrud/v1/audit/gateway/<fileId>[?jpath={jpathQueryString}]	Accept="application/json" or "application/xml" (default is "application/json")

Audit Log "Unstructured" Entry document retrieval, large file download	GET	/ecrud/v1/audit/fs/<fileId>	Accept header is ignored
Generic "Structured" Collection document retrieval	GET	/ecrud/v1/<dbName>/<collectionName>/<fileId>[?jpath={jpathQueryString}]	Content-Type="application/json" or "application/xml" or "plain/text" or "application/x-www-form-urlencoded" or "*/*" Accept="application/json" or "application/xml" (default is "application/json")
Generic Document Query On A "Searchable" Collection, with optional parameters	GET	/ecrud/v1/<dbName>/<collectionName>[?query={queryString}][&limit={maxRecords}&fields={columnProjections}&sort={sortCriteria}]	Accept="application/json" or "application/xml" (default is "application/json")
Generic Document Query On A GridFS "fs.files" Collection, with optional parameters	GET	/ecrud/v1/<dbName>/fs.files[?query={queryString}][&limit={maxRecords}&fields={columnProjections}&sort={sortCriteria}]	Accept="application/json" or "application/xml" (default is "application/json")
Generic Direct-To- GridFS document retrieval, large file download	GET	/ecrud/v1/<dbName>/fs/<fileId>	Accept header is ignored
Generic Update Entire-Document- Type, Single Document Operation	PUT	/ecrud/v1/<dbName>/<collectionName>/<fileId>	Content-Type="application/json" or "application/xml" or "plain/text" or "application/x-www-form-urlencoded"; document in request body contains <i>only</i> "field:value expressions" * (Note, document in request body may not contain MongoDB "Update Operators" **).

Generic Update-Entire- Document-Type, Multiple Documents Operation In A “Searchable” Collection	N/A	Not Supported	N/A
Generic Update-Specific-Documents Fields-Only-Type, Multiple Documents Operation In A “Searchable” Collection	PUT	Not Supported	N/A
Update-Entire-Documents-Type Operation In A GridFS Collection	N/A	Not Supported	N/A
Generic Delete a “Searchable” Collection Document	DELETE	<a href="/ecrud/v1/<dbName>/<collectionName>/<fileId>">/ecrud/v1/<dbName>/<collectionName>/<fileId>	Accept: application/json, application/xml
Generic Bulk Delete “Searchable” Collection Documents	DELETE	<a href="http(s)://{host}:{port}/ecrud/v1/<dbName>/<collectionName>[?query={queryString}]">http(s)://{host}:{port}/ecrud/v1/<dbName>/<collectionName>[?query={queryString}]	Accept: application/json, application/xml
Generic Delete GridFS Collection Document	DELETE	<a href="/ecrud/v1/<dbName>/fs/<fileId>">/ecrud/v1/<dbName>/fs/<fileId>	Accept: application/json, application/xml

* For more on “field:value expressions” for the Generic Update-Entire-Documents-Type Operation In A “Searchable” Collection, see the MongoDB Manual website (Reference 5), “Update” parameter and “Replace All Fields” sections at:

<http://docs.mongodb.org/manual/reference/method/db.collection.update/#db.collection.update>

**For more on “update operation expressions” for the Generic Update-Specific-Documents Fields-Only-Type Operation In A “Searchable” Collection, see the MongoDB Manual website (Reference 5), “Update” parameter and “Update Specific Fields” sections at:

<http://docs.mongodb.org/manual/reference/method/db.collection.update/#db.collection.update>

See also “Update Operators” page at:

<http://docs.mongodb.org/manual/reference/operator/update/>

Appendix B – eCRUD HTTP Request Payload Examples

The following sections represent example request body documents for several operations performed on the eCRUD Service.

Create/POST Request STR Subscription JSON Document

Request URL Pattern:

POST [http\(s\)://{host}:{port}/ecrud/v1/core/serviceTreatmentRecords.subscriptions](http(s)://{host}:{port}/ecrud/v1/core/serviceTreatmentRecords.subscriptions)

Required Request Header(s):

Content-Type: “application/json”

Example Request Payload:



subscriptionDocument
.json

Create/POST Request To “transform” Operation Request CAPRI DBQ XML Document

Request URL Pattern:

POST [http\(s\)://{host}:{port}/ecrud/v1/core/disabilityBenefitsQuestionnaires/transform](http(s)://{host}:{port}/ecrud/v1/core/disabilityBenefitsQuestionnaires/transform)

Required Request Header(s):

Content-Type: “multipart/form-data”;

Content-Desc: “niem/xml”

Required Multipart form requirements in request body:

- “part” is first and only part in multipart request;
- “part” content-type attribute is set, i.e. “application/xml”;
- “part” is in form element with a name of “file”;
- “part” has original filename stored in above “file” element.

Example Request Payload:



dbq.xml

Create/POST Request To “transform” Operation Request eCFT XML Document

Request URL Pattern:

POST [http\(s\)://{host}:{port}/ecrud/v1/core/electronicCaseFiles/transform](http(s)://{host}:{port}/ecrud/v1/core/electronicCaseFiles/transform)

Required Request Header(s):
Content-Type: "multipart/form-data";
Content-Desc: "niem/xml"

Required Multipart form requirements in request body:

- "part" is first and only part in multipart request;
- "part" content-type attribute is set, i.e. "application/xml";
- "part" is in form element with a name of "file";
- "part" has original filename stored in above "file" element.

Example Request Payload:



eCFTCaseFile -
XRay.xml

Create/POST Request STR Metadata XML Document

Request URL Pattern:
POST [http\(s\)://{host}:{port}/ecrud/v1/core/serviceTreatmentRecords](http(s)://{host}:{port}/ecrud/v1/core/serviceTreatmentRecords)

Required Request Header(s):
Content-Type: "application/xml"

Example Request Payload:



strMetadataXmlDoc.x
ml

Create/POST Request STR PDF Document With Large File Upload Request

Request URL Pattern:
POST [http\(s\)://{host}:{port}/ecrud/v1/core/fs](http(s)://{host}:{port}/ecrud/v1/core/fs)

Required Request Header(s):
Content-Type: "multipart/form-data"

Required Multipart form requirements in request body:

- "part" is first and only part in multipart request;
- "part" content-type attribute is set, i.e. "application/pdf";
- "part" is in form element with a name of "file";
- "part" has original filename stored in above "file" element.

Request Payload:

Binary stream of application/pdf sent in "multipart/form-data" request. This is usually handled by the eCRUD Consumer, i.e. "User Agent".

Create/POST Request “Structured audit log” JSON Document

Request URL Pattern:

POST [http\(s\)://{host}:{port}/ecrud/v1/audit/gateway](http(s)://{host}:{port}/ecrud/v1/audit/gateway)

Required Request Header(s):

Content-Type: “application/json”

Request Payload:

See DAS Client Gateway Technical Story.

Create/POST Request “Unstructured audit log” Document With Large File Upload Request

Request URL Pattern:

POST [http\(s\)://{host}:{port}/ecrud/v1/audit/fs](http(s)://{host}:{port}/ecrud/v1/audit/fs)

Required Request Header(s):

Content-Type: “multipart/form-data”

Required Multipart form requirements in request body:

- “part” is first and only part in multipart request;
- “part” content-type attribute is set, i.e. “application/xml”;
- “part” is in form element with a name of “file”;
- “part” has original filename stored in above “file” element.

Request Payload:

See DAS Client Gateway Technical Story.

Appendix C – eCRUD HTTP Response Headers

Table 9 -- eCRUD Response HTTP Response Headers

Response HTTP Header Name	Description	Example Value(s)
Connection	Options that are desired for the connection	keep-alive
Content-Disposition	Included in an eCRUD Response when a binary file attachment is retrieved from GridFS in response to a Read-direct request, used to suggest a default filename to the Consumer (See Reference 1, section 19.5.1 “Content-Disposition” for more information)	attachment; filename="fname.ext"
Content-Type	Content-type or MIME-type value of the response body returned from eCRUD Service	application/json, application/xml, text/plain
Content-Length	The length in octets (8-bit bytes) of the response body returned from eCRUD Service	176
Date	Date and time the response was created by eCRUD	Mon, 21 Apr 2014 21:50:04 GMT
ETag	ETag or entity-tag header used for caching to determine whether a given resource in the response has changed from a previous version	\\"184001143\\"

Appendix D – eCRUD Response Body Examples

The following sections represent example responses for several operations performed on the eCRUD Service.

Note: Except where noted, all of the response examples shown below are shown in the default content format of “application/json”. They can also be returned in the format of “application/xml” if an HTTP request “Accept” header is included and set to the value of “application/xml”.

Read-direct/GET Single DBQ “Attachment” Document (e.g. .pdf) From GridFS Collection Response

Request URL pattern:

GET <http://{host}:{port}/ecrud/v1/core/fs/<fileId>>

Example Request URL:

GET <http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/fs/535ec64d231f8d8318000008>

Example response with HTTP response headers, with .pdf binary response body translated to text:



icd-6-1.txt

Note: the original filename is available in the “Content-Disposition” HTTP response header.

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Read-direct/GET Single DBQ Metadata Document From “Searchable” Collection Response

Request URL pattern:

GET <http://{host}:{port}/ecrud/v1/core/disabilityBenefitsQuestionnaires/<fileId>>

Example Request URL:

GET

<http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/disabilityBenefitsQuestionnaires/535ec64e231f8d831800000a>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



icd-6-2.txt

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Read-direct/GET Single STR Subscription Document From “Searchable” Collection Response

Request URL pattern:

GET <http://{host}:{port}/ecrud/v1/core/serviceTreatmentRecords.subscriptions/<fileId>>

Example Request URL:

GET <http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords.subscriptions/535acf00e00d565d64000001>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Read-direct/GET Single STR Metadata Document From “Searchable” Collection Response

Request URL pattern:

GET <http://{host}:{port}/ecrud/v1/core/serviceTreatmentRecords/<fileId>>

Example Request URL:

GET
<http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords/535ad1816ff4ba5f64000003>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Read-direct/GET Single STR (e.g. .pdf) Document From GridFS Response

Request URL pattern:

GET <http://{host}:{port}/ecrud/v1/core/fs/<fileId>>

Example Request URL:

GET <http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/fs/535ad1806ff4ba5f64000001>

Example response with HTTP response headers, with .pdf binary response body text removed:



Note: the original filename is available in the “Content-Disposition” HTTP response header.

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Read-direct/GET Single eCFT (e.g. .pdf) Document From GridFS Collection Response

Request URL pattern:

GET <http://{host}:{port}/ecrud/v1/core/fs/<fileId>>

Example Request URL:

GET <http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/fs/536bed1e22ea6c8a32000008>

Example response with HTTP response headers, with .pdf binary response body translated to text:



icd-6-6.txt

Note: the original filename is available in the “Content-Disposition” HTTP response header.

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Read-direct/GET Single eCFT Metadata Document From “Searchable” Collection Response

Request URL pattern:

GET <http://{host}:{port}/ecrud/v1/core/electronicCaseFiles/<fileId>>

Example Request URL:

GET

<http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/electronicCaseFiles/536bed1f22ea6c8a3200000a>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



icd-6-7.txt

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Read-direct/GET Single Document From “audit/gateway” Collection Response

Request URL pattern:

GET <http://{host}:{port}/ecrud/v1/audit/gateway/<fileId>>

Example Request URL:

GET <http://bhietestsapp4.vaco.va.gov/ecrud/v1/audit/gateway/536513da727b31b273000005>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



icd-6-8.txt

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Read-direct/GET Single Document From “audit/fs” Collection Response

This example shows a retrieved audit log response from the GridFS “fs” collection in the audit database. The example shows a response to a GET request for an “original” DBQ document, with the embedded attachment left in the document. The response contained the document in “application/xml” format, so the audit document shows the “application/xml” format.

Request URL pattern:

GET <http://{host}:{port}/ecrud/v1/audit/fs/<fileId>>

Example Request URL:

GET <http://bhietestsapp4.vaco.va.gov/ecrud/v1/audit/fs/536cb69cc2293f3a24000009>

Example response with HTTP Response Headers shown, in “application/xml” formatted content:



icd-6-9.txt

Note: the original filename is available in the “Content-Disposition” HTTP response header.

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Read-Direct/GET Single Record From “fs.files” GridFS Metadata Collection Response

Request URL pattern:

GET <http://{host}:{port}/ecrud/v1/audit/fs.files/<fileId>>

Example Request URL:

GET <http://bhietestsapp4.vaco.va.gov/ecrud/v1/audit/fs.files/536cb69cc2293f3a24000009>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



icd-6-10.txt

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Read-Query/GET Multiple Documents From “Searchable” Collection Responses

1. Example query using no parameters, so that all documents are selected in a “searchable” collection:

Note that the default response size will define the upper limit of the number of documents returned in this case.

Request URL Pattern:

GET <http://{host}:{port}/ecrud/v1/<dbName>/<collectionName>>

Example Request URL:

GET <http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

2. Example query using “query=” parameter with a single selection criterion to select specific documents in a “searchable” collection:

Request URL Pattern:

GET <http://{host}:{port}/ecrud/v1/<dbName>/<collectionName>?query={queryString}>

Example Request URL:

GET [http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords?query={\"str:ServiceTreatmentRecord.str:CommonData.nc:Document.nc:DocumentStatus.nc:StatusText\":\"Completed\"}](http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords?query={\)

Example Request URL Encoded:

GET <http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords?query=%7b%22str:ServiceTreatmentRecord.str:CommonData.nc:Document.nc:DocumentStatus.nc:StatusText%22:%22Completed%22%7d>

<http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords?query=%7b%22str:ServiceTreatmentRecord.str:CommonData.nc:Document.nc:DocumentStatus.nc:StatusText%22:%22Completed%22%7d>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

3. Example query using “query=” parameter with multiple selection criteria to select specific documents in a “searchable” collection:
Note: This example query string uses the “\$regex” operator with its optional “\$options” operator.

Request URL Pattern:

GET <http://{host}:{port}/ecrud/v1/<dbName>/<collectionName>?query={queryString}>

Example Request URL:

GET

[http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords?query={\"str:ServiceTreatmentRecord.str:CommonData.nc:Person.nc:PersonName.nc:PersonGivenName\":{\"\\$regex\":\"Bettie\", \"\\$options\":\"i\"}\", \"str:ServiceTreatmentRecord.str:CommonData.nc:Person.nc:PersonName.nc:PersonSurName\":{\"\\$regex\":\"Bbtest\", \"\\$options\":\"i\"}}](http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords?query={\)

Example Request URL Encoded:

GET

<http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords?query=%7b%22str:ServiceTreatmentRecord.str:CommonData.nc:Person.nc:PersonName.nc:PersonGivenName%22:%7b%22%24regex%22:%22Bettie%22,%22%24options%22:%22i%22%7d,%22str:ServiceTreatmentRecord.str:CommonData.nc:Person.nc:PersonName.nc:PersonSurName%22:%7b%22%24regex%22:%22Bbtest%22,%22%24options%22:%22i%22%7d%7d>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



icd-6-11-3.txt

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

4. Example query using “fields=” parameter to select a subset of content from each document in a “searchable” collection:

Request URL Pattern:

GET <http://{host}:{port}/ecrud/v1/<dbName>/<collectionName>?fields={columnProjections}>

Example Request URL:

GET [http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords?fields={\"str:ServiceTreatmentRecord.str:CommonData.nc:Person\":1}](http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords?fields={\)

Example Request URL Encoded:

GET <http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords?fields=%7b%22str:ServiceTreatmentRecord.str:CommonData.nc:Person%22:1%7d>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



icd-6-11-4.txt

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Read-Query/GET Multiple Records From “fs.files” GridFS Metadata Collection Response

Request URL Pattern:

GET <http://{host}:{port}/ecrud/v1/<dbName>/fs.files>

Example Request URL:

GET <http://bhietestsapp4.vaco.va.gov/ecrud/v1/core/fs.files>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



icd-6-12.txt

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Create/Update Single Document In “Searchable” Collection Success Response

Request URL Pattern:

POST or PUT <http://{host}:{port}/ecrud/v1/<dbName>/<collectionName>>

Response body pattern, in “application/json” formatted content:

```
{
  -document: {
    id: "$ObjectID(<fileIdOfFileCreatedOrUpdated>)"
    path:
      "/ecrud/v1/<dbName>/<collectionName>/<fileIdOfFileCreatedOrUpdated>"
    uploadDate: "$Date(<ISODatetimeFileWasCreatedOrUpdatedInVDS>)"
  }
}
```

Example Request URL:

POST or PUT <http://bhietestsapp4.vaco.va.gov/ecrud/v1/core/disabilityBenefitsQuestionnaires>

Status Code: 200 OK

Example Response HTTP Headers:

Content-Length: 27538

Content-Type: application/json

Date: Mon, 12 May 2014 16:17:42 GMT

Etag: "662265732"

Server: Jetty(9.1.3.v20140225)

X-Powered-By: Express

x-response-time: 177ms

Example response body shown, in “application/json” formatted content:


```
{
  -document: {
    id: "$ObjectID(535e9a85e3da922d0c000001)"
    path:
      "/ecrud/v1/core/disabilityBenefitsQuestionnaires/535e9a85e3da922d0c000001"
    uploadDate: "$Date(2014-04-28T18:14:29.490Z)"
  }
}
```

Note: For Update/PUT requests only: the request body must contain an “_id” element at the root level which matches the “<fileId>” in the request URL, and a new, valid “uploadDate” element at the root level.

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Create Single Document In GridFS Collection Success Response

Request URL Pattern:

POST <http://{host}:{port}/ecrud/v1/<dbName>/fs>

URL response body pattern, in “application/json” formatted content:

```
{
  "file": {
    "type": "<MIME-typeOfStoredFile-FromMultipartFormContent-typeAttribute>",
    "id": "$ObjectID(<fileId>)",
    "size": <integerInBytes>,
    "root": "fs",
    "path": "filename.extension",
    "lastModified": "$Date(<ISODateTimeFileWasCreatedOrUpdatedInVDS>)"
  }
}
```

Example Request URL:

POST <http://bhietestsapp4.vaco.va.gov/ecrud/v1/core/fs>

where:

- (first and only) multipart “part”’s form file name is “file”,
- “file” above has the value of “dbq.xml”,
- the “part” has a content-type attribute value of “application/xml”,
- and the total request has the HTTP Request “Content-Type” value of “multipart/form-data”:

Example response with HTTP Response Headers shown, in “application/json” formatted content:

```
{
  "file": {
    "type": "application/xml",
    "id": "$ObjectID(5363fb3066427a600c000007)",
    "size": 212959,
    "root": "fs",
    "path": "dbq.xml",
```

```
"lastModified": "$Date(2014-05-02T20:08:16.793Z)"
}
```

Note: Update/PUT is not currently supported for GridFS by eCRUD.

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Delete Single Document In Searchable Collection Success Response

Request URL Pattern:

DELETE <http://{host}:{port}/ecrud/v1/<dbName>/<collectionName>/<fileId>>

Example Request URL:

DELETE

<http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords.subscriptions/535acf00e00d565d64000001>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



icd-6-15.txt

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Delete Multiple Documents In Searchable Collection Success Response

Request URL Pattern:

DELETE [http://{host}:{port}/ecrud/v1/<dbName>/<collectionName>\[?query={queryString}\]](http://{host}:{port}/ecrud/v1/<dbName>/<collectionName>[?query={queryString}])

Example Request URL:

DELETE

<http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords.subscriptions>

Example response with HTTP Response Headers shown, in “application/json” formatted content:



icd-6-16.txt

Note: The example above does not have the optional “query=” selection parameter, but selects EVERY document contained within the example collection for deletion.

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Delete Single Document In GridFS Collection Success Response

Request URL Pattern:

DELETE <http://{host}:{port}/ecrud/v1/<dbName>/fs/<fileId>>

Example Request URL:

DELETE <http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/fs/5278ed1137f4d5441b000019>

Request Example Headers:

Similar to “Section 6.15 - Delete Single Document In Searchable Collection Success Response”.

Example response with HTTP Response Headers shown, in “application/json” formatted content:
Body:

```
{
  "success": true
}
```

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Example Operation Failure Response (using 400/Bad Request response as an example)

1. For POST to “custom” “transform” URL operations, when uploaded multipart “part” file’s form name is not “file”:

Response Headers:

Status: 400 Bad Request

Content-Type: “application/json”

Response body:

```
{
  Error: '400 - name="file" is required'
}
```

2. Default error response for PUT request (with <fileId> in request URL):

Response Headers:

Status: 400 Bad Request

Content-Type: “application/json”

Response Body:

```
{
  Error: '400 - a valid uploadDate is required'
}
```

3. Error response for wrongly-formatted or missing “uploadDate” in request body for PUT request (with <fileId> in request URL):

Response Headers:

Status: 400 Bad Request

Content-Type: “application/json”

Response Body:

```
{
  Error: '400 - a valid uploadDate is required'
}
```

}

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Example Operation Failure Response (using 415/UnsupportedMediaType response as an example)

1. For POST to “custom” “transform” URL operations, when a request with an unsupported “Content-Type” Response Header value is received:

Request Headers:

Status: 415 Unsupported Media Type

Content-Type: “application/json”

Response Body:

```
{
    '415 – Content-Type: <MimeTypeValueFromRequestContent-TypeHeader> is
not supported'
}
```

2. For POST to “custom” “transform” URL operations, when a request with an unsupported “Content-Desc” Response Header value is received:

Request Headers:

Status: 415 Unsupported Media Type

Content-Type: “application/json”

Response Body:

```
{
    '415 – Content-Desc: <MimeTypeValueFromRequestContent-DescHeader> is
not supported'
}
```

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Example Operation Failure Response (using 404/Not Found response as an example)

1. Default For GET-Direct “404/Not Found” failure response:

Response Headers:

Status: 404 Not Found

Content-Type: “text/plain”

Response Body:

“Not Found”

2. For GET-Query “404/Not Found” failure response for non-existent collection:

Response Headers:

Status: 404 Not Found
Content-Type: "application/json"

Response Body:
{0}

3. For GET-Query "404/Not Found" failure response for 'no matching records found' condition:

TBD

4. For POST "404/Not Found" error response:

Response Headers:
Status: 404 Not Found
Content-Type: "text/html"

Response Body:
"Cannot POST /ecrud/v1/<dbName>/<collectionName>/<fileId>"

5. For DELETE "404/Not Found" failure response:

TBD

Note: "application/xml" can also be used in the "Accept" request header, and will cause the response to be returned from eCRUD in the "application/xml" Content-Type format.

Example Operation Error Response (using 500/Internal Server Error as an example)

TBD

Create/POST STR Metadata Document Success Response

Request URL Pattern:

POST <http://{host}:{port}/ecrud/v1/core/serviceTreatmentRecords>

Example Request URL:

POST <http://bhietestsapp4.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords>

Example Response Headers:

Similar to those in "6.13 - Create/Update Single Document In "Searchable" Collection Success Response".

Example response body shown, in "application/json" formatted content:

```
{
  -document: {
    id: "$ObjectID(535e9a85e3da922d0c000001)"
    path: "/ecrud/v1/core/serviceTreatmentRecords/535e9a85e3da922d0c000001"
    uploadDate: "$Date(2014-04-28T18:14:29.490Z)"
  }
}
```

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Create/POST STR Subscription Document Success Response

Request URL Pattern:

POST <http://{host}:{port}/ecrud/v1/core/serviceTreatmentRecords.subscriptions>

Example Request URL:

POST <http://bhietestsapp4.vaco.va.gov/ecrud/v1/core/serviceTreatmentRecords.subscriptions>

Example Response Headers:

Similar to those in “6.13 - Create/Update Single Document In “Searchable” Collection Success Response”.

Example response body shown, in “application/json” formatted content:

```
{
  -document: {
    id: "$ObjectID(535e9a85e3da922d0c000001)"
    path:
      "/ecrud/v1/core/serviceTreatmentRecords.subscriptions/535e9a85e3da922d0c000001"
    uploadDate: "$Date(2014-04-28T18:14:29.490Z)"
  }
}
```

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Create/POST CAPRI And Other Partner DBQ XML Document With “transform” Operation URL Success Response

Request URL Pattern:

POST <http://{host}:{port}/ecrud/v1/core/disabilityBenefitsQuestionnaires/transform>

Response body pattern, in “application/json” formatted content:

```
{
  {<documentWhichWasCreatedInSearchableCollectionInVDS>
  },
  "uploadDate": "$Date(<ISODateTimeFileWasCreatedInSearchableCollectionInVDS>)",
  "_id": "$ObjectID(<fileIdOfFileCreatedInGivenSearchableCollection>)"
}
```

Example Request URL:

POST

<http://bhiedevappch3.vaco.va.gov/ecrud/v1/core/disabilityBenefitsQuestionnaires/transform>

Example Response Headers:

Similar to those in “6.13 - Create/Update Single Document In “Searchable” Collection Success Response”.

Example response, in “application/json” formatted content:

Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Create/POST eCFT XML Document With “transform” Operation URL Success Response

Request URL pattern:

POST <http://{host}:{port}/ecrud/v1/core/electronicCaseFiles/transform>

Response body pattern, in “application/json” formatted content:

```
{
  {<documentWhichWasCreatedInSearchableCollectionInVDS>
},
  "uploadDate": "$Date(<ISODateTimeFileWasCreatedInSearchableCollectionInVDS>)",
  "_id": "$ObjectID(<fileIdOfFileCreatedInGivenSearchableCollection>)"
}
```

Example Request URL:

POST <http://bhietestsapp4.vaco.va.gov/ecrud/v1/core/electronicCaseFiles/transform>

Example Response Headers:

Similar to those in “6.13 - Create/Update Single Document In “Searchable” Collection Success Response”.

Example response, in “application/json” formatted content:



Note: “application/xml” can also be used in the “Accept” request header, and will cause the response to be returned from eCRUD in the “application/xml” Content-Type format.

Appendix E – JSON to BSON Decoration Capability

This appendix describes the supported BSON decoration capability available in eCRUD as of DAS release v.4.4.4.

The BSON decorations allow some elements within JSON documents stored in MongoDB to be represented in certain MongoDB-specific, binary datatypes instead of only string types to allow for greater ability to query for documents.

eCRUD has functionality to accept specific text items to incoming JSON documents so that the MongoDB database will convert them from a string type into the specified binary type. There are two types of decorations supported at present used to convert items embedded inside new JSON documents stored: 1) the “\$ObjectID(...)” decoration; 2) and the “\$Date (...)” decoration. These decorations are stored in the JSON documents in the VLER Data Store, and will need to first be removed from documents which are retrieved from the VLER Data Store if these documents need to be validated, and if the original data integrity needs to be preserved.

The “\$ObjectID(...)” decoration is added to BSON “_id” values embedded inside of JSON documents before they are sent in a Create or Update request to eCRUD, so that they are represented as the BSON “ObjectId” type in the VLER Data Store. This is an example of a JSON document containing the “\$ObjectID(...)” decoration for an embedded document _id string value:

```
{ "MyFileId": "$ObjectID(53078d5170c21bd75e000001)" }
```

where “53078d5170c21bd75e000001” is a unique hex string identifier of an “ObjectId” MongoDB datatype.

Note that the “_id” element either stored or created automatically by MongoDB for each document is stored as the BSON ObjectId type. (When no “_id” value is passed in the document to be stored, MongoDB creates one with a UUID automatically).

Consumers can then use the “queryString” JSON value in all eCRUD URLs containing the “query=” parameter to query for embedded ObjectId values as BSON Objects, instead of only as strings. This is an example of a queryString URL with the \$ObjectID() decorator:

[http://myEcrudHost:portNum/ecrud/v1/core/myCollectionName?query={ \"MyFileId\": \"\\$ObjectID\(53078d5170c21bd75e000001\)\" }](http://myEcrudHost:portNum/ecrud/v1/core/myCollectionName?query={\)

For more information on the ObjectId type, see the MongoDB API documentation at:

<http://mongodb.github.io/node-mongodb-native/api-bson-generated/objectid.html>

The “\$Date (...)” decoration is can be added to date string values with certain date formats inside of JSON documents before they are stored in a Create or Update request to eCRUD, so that they are represented as the BSON “Date” type in the VLER Data Store. The acceptable date formats are those listed in the date.js library, which is detailed here: <http://www.datejs.com/>, as well as the “ISO Date” value, e.g. "1924-12-03T05:00:00Z". Here is an example of a JSON document containing the “\$Date(...)” decoration for an several embedded string values:

```
{
  "Date 1" : "$Date(3-25-2014)",
  "DateTime Now" : "$Date(now)",
  "7 Days Ago" : "$Date(-7days)"
}
```


This allows these JSON date values to be represented in the BSON “Date” type defined by MongoDB. Note that the “uploadDate” elements created automatically by the eCRUD Service for each document when it is stored and it also stored in the BSON Date type.

Consumers can then use the “queryString” JSON value in all eCRUD URLs containing the “query=” parameter to query for Date values as dates, instead of as strings. This is an example of a queryString URL with the \$Date() decorator:

[http://myEcrudHost:portNum/ecrud/v1/core/myCollectionName?query={\"Date1\":{\"\\$gte\":{\"\\$Date\(-10days\)\"}}](http://myEcrudHost:portNum/ecrud/v1/core/myCollectionName?query={\)

For more information on the Date type, see the MongoDB API documentation at:
<http://docs.mongodb.org/manual/reference/method/js-constructor/>