CRDC Submission Portal APIs

1. Introduction

This document walks you through the basics of submitting data to CRDC Submission Portal using APIs. Information provide in CRDC Data Submission Instructions will not be included in this document. It is recommended to get familiar with CRDC Data Submission Instructions and submission workflow using CRDC Data Submission Portal before interacting with CRDC Submission APIs.

CRDC Submission Portal provides a set of Graphql APIs (https://graphql.org/). There is only one API endpoint, https://hub.datacommons.cancer.gov/api/graphql. Different functions are provided as separate API queries (read) and mutations (write). Users may use API tools like Postman as client to interact with the APIs. Users may also choose to write code in any popular programming languages to interact with the APIs.

All CRDC Submission Portal API queries and mutations use HTTP POST method. The APIs will always return HTTP code 200 (success) even when API call fails. The API returns data and errors in JSON format. If an API call succeeds, the returned data will be under "**data**" key. If any error occurs, the error information will be returned under "**error**" key ("**data**" key will be null).

CRDC Submission Portal API is authentication and authorization controlled. API tool (or your code) needs to be set up to send "Bearer Token" in "**Authorization**" header.

A Graphql API's schema contains definitions of all queries, mutations and their parameters and return types. This document will only provide information that cannot be found in the Graphql schema.

2. Prerequisites

- A valid API token downloaded from Data Submission Portal.
- API testing tool, such as Postman or GraphiQL or custom code.
- Knowledge of using API tools and/or interacting with APIs in code.
- Basic knowledge of Graphql APIs (https://graphql.org/).

3. Conventions

A dot separated property name is used to describe the hierarchy in the data. For example, **submissions._id** means **_id** property under **submissions** property. See screenshot below.

```
"total": 24,
"submissions": [
    {
        "_id": "de9f113c-1a9d-4740-9495-2c595976b440",
        "name": "DELETE-1025_Delete_1201",
        "submitterName": "crdc.g.submtr",
```


4. Starting a new submission

Before creating a data submission, user needs to determine which study the data submission will be submitted to. To retrieve approved studies for a user's organization, call API query **listApprovedStudiesOfMyOrganization**.

API signature:

```
listApprovedStudiesOfMyOrganization: [ApprovedStudy]
```

Important return values:

- **_id**: should be used as **studyID** parameter of **createSubmission** API.

To create a new data submission, call API mutation **createSubmission**.

API signature:

```
createSubmission (
    studyID: String!,
    dbGaPID: String,
    dataCommons: String!,
    name: String!,
    intention: String!,
    dataType: String!
): Submission
```

Important parameters:

- **studyID**: **_id** field of an approved study
- **dbGaPID**: required for a controlled access study
- **dataCommons:** data common's name such as "CDS", "ICDC" etc.
- **name**: a user selected name for the submission
- **intention**: should be one of ["New/Update", "Delete"]
- **dataType**: should be one of ["Metadata Only", "Metadata and Data Files"]

Important return values:

- **_id:** submission's ID, aka. submissionID

5. Continuing an existing submission

a. Retrieve a list of all submissions

To retrieve a list of all submissions a user has access to, call API query **listSubmissions.**

API signature:

```
listSubmissions(
  organization: String,
  status: String,
  first: Int = 10,
  offset: Int = 0,
  orderBy: String = "updatedAt",
  sortDirection: String = "DESC"
): ListSubmissions
```

Important parameters:

- **Status:** results will be filtered by this parameter if one or more of the following values is provided: ["New", "In Progress", "Submitted", "Released", "Completed", "Archived", "Canceled", "Rejected", "Withdrawn", "Deleted", “All”]. If “All” is provided, no filter will be applied.
- **first:** number of records to be returned, if -1 is sent, API will return all available data
- **offset:** skip given number of records before returning data
- **orderBy:** property name used to sort returned data
- **sortDirection:** should be one of [“ASC”, “DESC”]

Important return values:

- **submissions._id:** submission’s ID, aka. submissionID

b. Retrieve information about a submission

To retrieve detailed information about a submission, call API query **getSubmission**.

API signature:

```
getSubmission(_id: ID!): Submission
```

Important parameters:

- **_id**: submission’s ID, aka. submissionID

To retrieve statistics of a submission, call API query **submissionStats.**

API signature:

```
submissionStats(_id: ID!): SubmissionStats
```

Important parameters:

- **_id**: submission's ID, aka. submissionID

To retrieve uploaded metadata, call API query **getSubmissionNodes.**

API signature:

```
getSubmissionNodes(
  submissionID: String!,
  nodeType: String!,
  status: String = "All",
  nodeID: String,
  first: Int = 10,
  offset: Int = 0,
  orderBy: String = "nodeID",
  sortDirection: String = "ASC"
): SubmissionNodes
```

Important parameters:

- **submissionID**: submission's ID
- **nodeType**: type of the metadata node to be returned
- **status:** should be one of ["All", "New", "Error", "Passed", "Warning"]. If "All" is provided. no filter will be applied, otherwise, return will be filtered by metadata's status.
- **nodeID:** if provided, return will be filtered by provided node ID, any node ID partially match given value will be returned.
- **first:** number of records to be returned, if -1 is sent, API will return all available data
- **offset:** skip given number of records before returning data
- **orderBy:** property name used to sort returned data
- **sortDirection:** should be one of ["ASC", "DESC"]

Important return values:

- **IDPropName**: name of the metadata node's ID property
- **Properties**: names of all metadata node's properties
- **Nodes**.props: a JSON string contains all properties of the metadata node, needs to be parsed as JSON in the code.

c. Uploading Files and Manifests

It is recommended to upload data files via Uploader CLI Tool. It is also possible to upload data files by writing code.
To retrieve an CLI configuration file, call API query **retrieveCLIConfig**. Returned data contains correct content and format (including line breaks and indentation). It is recommended to call this API in code and save returned string into a YAML file without modifying the content in anyway. For example, "my-cli-configuration.yml". API tools have their own way of displaying data contains line breaks, it will be hard to preserve original return data in an API tool.

API signature:

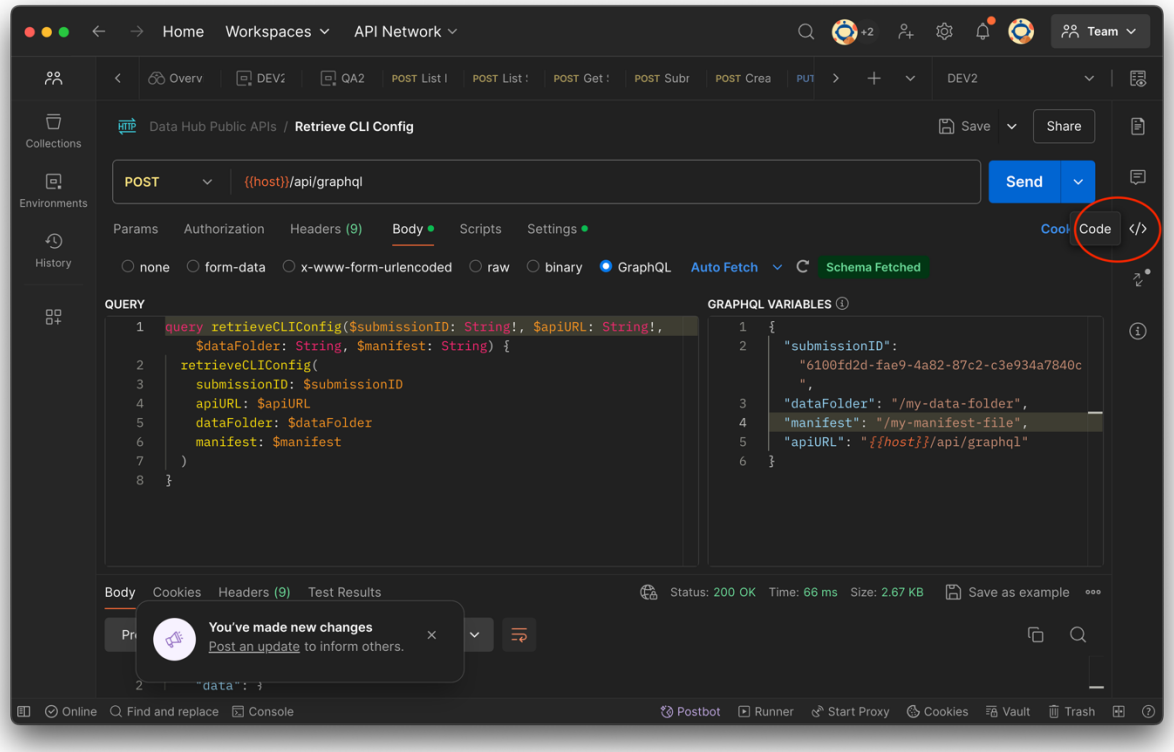
```
retrieveCLIConfig(
  submissionID: String!,
  apiURL: String!,
  dataFolder: String,
  manifest: String
): String
```

Important parameters:

- **submissionID**: submission's ID
- **apiURL**: must be **"https://hub.datacommons.cancer.gov/api/graphql"**
- **dataFolder**: local path of data files folder, for example *"/Users/me/my-data-files-folder"*
- **manifest**: local path of manifest file, for example *"/Users/me/my-metadata-folder/my-file-manifest.tsv"*

To save CLI configuration file retrieved as a file without writing code, follow steps below:

- Setup API request in Postman
- Click "Code" button (</>) on the right-side bar
- Copy and paste the command into a terminal (cmd on Windows), then add following code to end of the command " ***| jq -r '.["data"]["retrieveCLIConfig"]' > my-cli-configuration.yml***"



If uploading data files in code is preferred, user may call API mutation **createTempCredentials** to retrieve a set of temporary credentials to use in the code.

API signature:

```
createTempCredentials (submissionID: ID!): TempCredentials
```

Important parameters:

- **submissionID**: submission's ID

Metadata templates can be uploaded via Uploader CLI Tool. It is also possible to upload metadata templates via API. If using API is preferred, it is recommended to perform following steps in code rather than in API tool:

- Step 1: create a "batch" by calling API mutation **createBatch**.
- Step 2: upload metadata templates using pre-signed URLs retrieved in step 1.
- Step 3: update upload results by calling API mutation **updateBatch.** API will take the input as is and store the status of the files in the database. An asynchronous essential validation will be triggered by the API call, and the batch's status will be updated based on validation result. If validations passed, metadata will be loaded into submission database and status will be set to "**Uploaded**". Otherwise, batch status will be set to "**Failed**". Note, individual file status will be based on the input, and will not be updated by essential validation service.
- Step 4: retrieve essential validation results by calling API query **listBatches**

createBatch:

API signature:

```
createBatch (
  submissionID: ID!,
  type: String,
  files: [FileInput]
): NewBatch
```

Important parameters:

- **submissionID**: submission's ID
- **type:** should be one of ["metadata", "data file"]
- **files.size**: size of the file to be uploaded. This parameter is stored but not used, and will be removed in future releases.

Important return values:

- **_id**: batch's internal ID, aka batchID
- **files.signedURL**: S3 pre-signed URL that can be used to upload metadata templates

updateBatch:

API signature:

```
updateBatch (batchID: ID!, files: [UploadResult]): Batch
```

Important parameters:

- **batchID:** batch's internal ID
- **files.skipped:** reserved for CLI use, should set to false

Important return values:

- **_id**: batch's internal ID, aka batchID
- **displayID**: batch's UI ID
- **status**: current status of the batch. It will change after essential validation is finished (explained in previous sections). Please call **listBatches** to get the latest status.

To retrieve information about all batches, call API query **listBatches**.

API signature:

```
listBatches(
  submissionID: ID!,
  first: Int = 10,
  offset: Int = 0,
```

```
  orderBy: String = "updatedAt",
  sortDirection: String = "DESC"
): ListBatches
```

Important parameters:

- **submissionID**: submission's ID
- **first:** number of records to be returned, if -1 is sent, API will return all available data
- **offset:** skip given number of records before returning data
- **orderBy:** property name used to sort returned data
- **sortDirection:** should be one of ["ASC", "DESC"]

Important return values:

- **batches._id**: batch's internal ID, aka batchID
- **batches.displayID**: batch's UI ID
- **batches.files.nodeType:** node type contained in the metadata file. This value is only available when the metadata file can be successfully read by the essential validation service. Otherwise, it will be null.

d. Deleting data

To delete a metadata node, call API mutation **deleteDataRecords.**

API signature:

```
deleteDataRecords(
  submissionID: String!,
  nodeType: String!,
  nodeIDs: [String!]
): DataValidation
```

Important parameters:

- **submissionID**: submission's ID
- **nodeType**: type of the metadata node to be deleted, or "**data file**" if deleting data files from S3 bucket is desired.
- **nodeIDs:** a list of node IDs, API will delete metadata node that matches provided node IDs.

Important return values:

- **success**: a Boolean value indicates if a deletion operation has been successfully initialized, the deletion will be performed asynchronously.

To delete a data file from submission bucket, call API mutation **deleteDataRecords.**

API signature:

```
deleteDataRecords(
  submissionID: String!,
  nodeType: String!,
  nodeIDs: [String!]
): DataValidation
```

Important parameters:

- **submissionID**: submission's ID
- **nodeType**: must be "**data file**"
- **nodeIDs:** a list of file names to be deleted.

Important return values:

- **success**: a Boolean value indicates if a deletion operation has been successfully initialized, the deletion will be performed asynchronously.

e. Running validations

To validate uploaded data, call API mutation **validateSubmission**.

API signature:

```
validateSubmission(
  _id: ID!,
  types: [String]
  scope: String
): DataValidation
```

Important parameters:

- **_id**: submission's ID, aka. submissionID
- **types:** any combination of following values: ["metadata". "data file"]
- **scope:** should be one of ["New", "All"]

Important return values:

- **success**: a Boolean value indicates if a validation has been successfully initialized, it has no relationship to the validation's result.

To retrieve detailed validation issues, call API query **submissionQCResults.**

API signature:

```
submissionQCResults(
  _id: ID!,
  nodeTypes: [String],
  batchIDs: [ID],
  severities: String,
  first: Int = 10,
```

```
  offset: Int = 0
  orderBy: String = "uploadedDate",
  sortDirection: String = "DESC"
): QCResults
```

Important parameters:

- **_id**: submission's ID, aka. submissionID
- **nodeTypes:** a list of metadata node types or "data file", return will be filtered by given metadata node types or file validation results if "data file" is given.
- **batchIDs:** a list of batches' internal IDs, return will be filtered by given batch internal IDs
- **severities**: should be one of ["All", "Error", "Warning"] , return will be filtered by given issue severity. "All" means both errors and warnings.
- **first:** number of records to be returned, if -1 is sent, API will return all available data
- **offset:** skip given number of records before returning data
- **orderBy:** property name used to sort returned data
- **sortDirection:** should be one of ["ASC", "DESC"]

Important return values:

- **validationType**: either "metadata" or "data file"
- **submittedID:** a metadata node's ID, or file name of a data file

f. Submitting your Final Dataset

To submit a submission for review, call API mutation **submissionAction.**

API signature:

```
submissionAction (
  submissionID: ID!,
  action: String!
  comment: String
): Submission
```

Important parameters:

- **submissionID**: submission's ID
- **action:** should be "**Submit**" for this use case. Valid value includes ["Submit", "Withdraw", "Cancel"]