BRICS USER GUIDE

Query Tool API

QUERY TOOL APPLICATION PROGRAMMING INTERFACE

The Query Tool Application Programming Interface (API) provides a RESTful service that enables users to access Query Tool functionality through HTTPS requests. These API endpoints can be accessed through any programming language or tool that allows HTTPS requests, such as curl, Python, or R.

Objective

The objective of this User Guide is to demonstrate how to use the Query Tool API. Specifically, this guide will cover the following items:

- Obtaining an API token through a BRICS Portal Site
- Extending an API session
- Demonstrating how endpoint requests should be formatted
- Providing an overview of the functionality this API provides

For detailed information about each endpoint in the Query Tool API, please access the Swagger API Documentation for the BRICS Instance you are interested (linked below). These pages provide information about each API endpoint in the OpenAPI format.

BRICS Instance	Query API
FITBIR	https://fitbir.nih.gov/gateway/query-api/swagger-ui/index.html#/
NEI	https://brics.nei.nih.gov/gateway/query-api/swagger-ui/index.html#/
PDBP	https://pdbp.ninds.nih.gov/gateway/query-api/swagger-ui/index.html#/

Table 1 - Query API Swagger Pages

To see examples on how to call the endpoints from Python, please refer to the accompanying Jupyter Notebook.

Authentication – API Token

To access the Query Tool API, a user must provide an API token with each request.

The API token serves as your identity and access control to the entire system. This token is like a password and should be protected as such. Each token expires after 30 minutes, but a valid token can be used to obtain a new token to extend the user's session. For each session, the initial token must be obtained through the Portal site as described below.

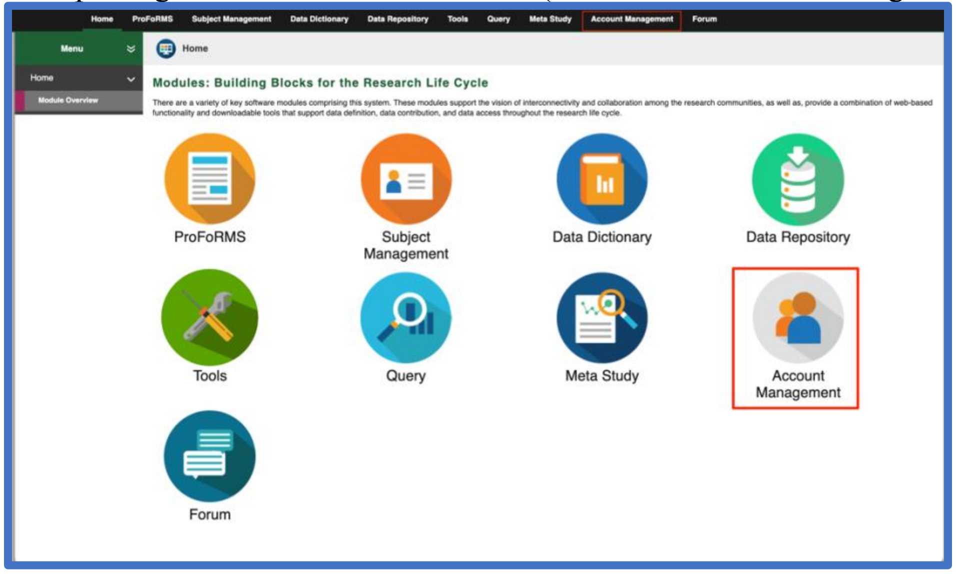
There are two aspects of BRICS API Tokens that are worth noting:

1. Extending a session and retrieving a new token does not cause the initial token to be invalidated. Both the new token and old token will be valid until their respective expiration times.
2. Tokens on the same session are all affected if the user logs out. The API token retrieved from the Portal is associated with the user's browser session. If the user logs out (or times out) on the browser, then all tokens in this session will be invalidated. Users can obtain a token on a new, API-only session by following the steps outlined in the section *Obtaining an API-Only Token*.

Retrieving an API Token from the Portal Site

Users can retrieve an API token from the Portal Site by following these steps:

1. Log into your BRICS instance and select the Account Management module from either the top navigation bar or the module icon (both outlined in red rectangles in Figure 1).

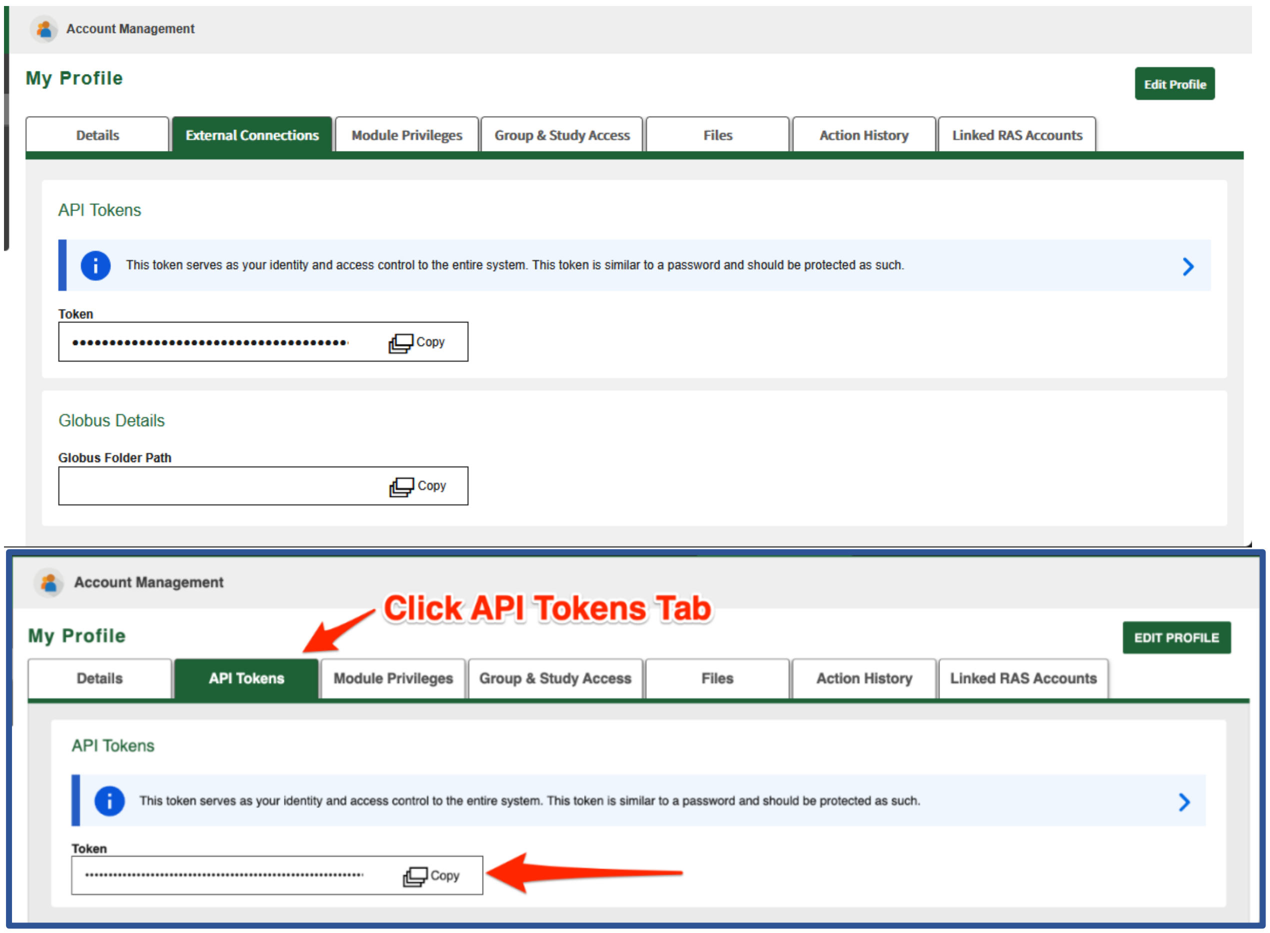


Modules: Building Blocks for the Research Life Cycle

There are a variety of key software modules comprising this system. These modules support the vision of interconnectivity and collaboration among the research communities, as well as, provide a combination of web-based functionality and downloadable tools that support data definition, data contribution, and data access throughout the research life cycle.

Figure 1 - Portal Workspace with Account Management links outlined in red rectangles

2. Once the user's profile page ("My Profile") loads, click the **External Connections** tab, find the text box contained in the "API Tokens" section, then click the "Copy" button to copy the token to your clipboard.



API Tokens

This token serves as your identity and access control to the entire system. This token is similar to a password and should be protected as such.

Token

Globus Details

Globus Folder Path

API Tokens

This token serves as your identity and access control to the entire system. This token is similar to a password and should be protected as such.

Token

Figure 2 - API Tokens page on a user's profile page

Using the API Token

When making calls to any BRICS API endpoint, include this token with the prefix "Bearer" as the value for HTTPS header "Authorization". Explicitly, the name of the header will be **Authorization** and the value will be **Bearer *API TOKEN*** For example:

Authorization: Bearer ABC.ABC123.123

Note the following potential pitfalls in providing the token in the Authorization header:

a. The capital "A" for the "Authorization" header.
b. The capital "B" in the prefix "Bearer".
c. The single space between "Bearer" and the token itself.

The token itself is a JSON Web Token (JWT) which complies with RFC 7519 and useful information, such as its expiration time, can be extracted from it.

Obtaining an API-Only Token

The API token obtained through the portal site is tied to the browser session. This means that if the browser session gets logged out (e.g. due to inactivity), this API token will be invalidated (as will any token obtained by extending this session). Users are able to use this token to get an "API-only" that is not tied to any session.

An API-only token can be obtained by making a HTTPS request with the following format:

GET /gateway/rasauth/authorize/apiOnlyToken
Host: <BRICS Instance's Domain>
Authorization: Bearer < Your access token >

For example, on NEI a curl request to get an API-only token would like:

```
curl -X GET -H 'Authorization: Bearer <YOUR-TOKEN>'
https://brics.nei.nih.gov/gateway/rasauth/user/apiOnlyToken
```

Extending the API Session by Getting a New Token

To renew the token, the following conditions must be met:

1. The current token must not be expired. See the "exp" claim within the token for its expiration time as a Unix time stamp.
2. The session associated with this token must not have been logged out.
3. The underlying login session must not be more than 15 days old.

To renew the token, make a webservice call with the following format:

GET /gateway/rasauth/user/extendApi
Host: <BRICS Instance's Domain>
Authorization: Bearer < Your access token >

For example, on NEI a curl request to extend the session token would like:

```
curl -X GET -H 'Authorization: Bearer <YOUR-TOKEN>'
https://brics.nei.nih.gov/gateway/rasauth/user/extendApi
```

Query Tool API Endpoints

This section briefly covers of the endpoints available in the Query Tool API. For a few example endpoints, it shows what each endpoint does, how to call them, and provides minimal examples in Python (using the popular *requests* library) showing how to use these endpoints.
Please refer to the accompanying Jupyter notebook for fuller examples of each of these endpoints.

As mentioned in the Authentication section, a token is needed to retrieve data for all subsequent endpoints.

The base url for each method is the *gateway/query-api* for your BRICS instance. For example, on NEI, the base url is *https://brics.nei.nih.gov/gateway/query-api*. Each API request should be directed to the specified path on the base URL and should use the method listed in the documentation.

Study API

The Query Tool Study API provides the following endpoints. In the following descriptions, the term *demographic information* refers to a list of number of subjects broken down by 10-year age-ranges and sex.

HTTPS Method	**Path**	**Description**
GET	/study	Get information about one or more studies.
GET	/study/form	Get information about studies that have data for the provided form structure(s).
GET	/study/detailed	Retrieve information about the study along with its demographic information.
GET	/study/detailedFull	Retrieve information about the study along with overall demographic information as well as demographic information about each form structure.
POST	/study/detailed/form	Retrieve information about the study along with demographic information for only the provided form structure.

Two example endpoints are documented below.

Getting All Study Information

This service will return information about each study that has data in the instance. Optionally, the user can select which studies to retrieve this information for by providing one or more prefixed Study I Ds[1] to the parameter *prefixedId*.

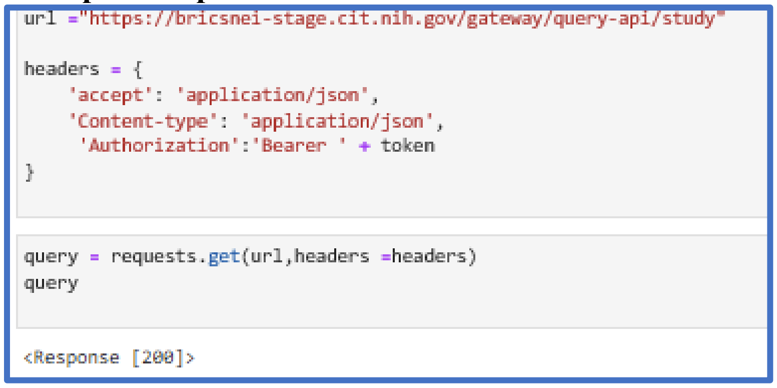
Endpoint URL: https://brics.nei.nih.gov/gateway/query-api/study

Method	GET
Headers	accept: application/json Authorization: Bearer + Token
Parameters	Optional: prefixedId =

Sample Python request	`response = requests.get(`
	`  "https://brics.nei.nih.gov/gateway/query-api/study",`
	`  headers = headers`
	`);`
	
	`response = requests.get(`
	`"https://brics.nei.nih.gov/gateway/queryapi/form/study?prefixedId=INSTANCE-STUDY0000001",`
	`  headers = headers`
	`);`

[1] A prefixed study ID is in the format <INSTANCE-NAME>-STUDY<7-DIGIT-ID>. For example, NEI study 211 has a prefixed id of NEI_BRICS-STUDY0000211.

Example of Input:



```
url ="https://bricsnei-stage.cit.nih.gov/gateway/query-api/study"

headers = {
    'accept': 'application/json',
    'Content-type': 'application/json',
     'Authorization':'Bearer ' + token
}
```

```
query = requests.get(url,headers =headers)
query
```

```
<Response [200]>
```

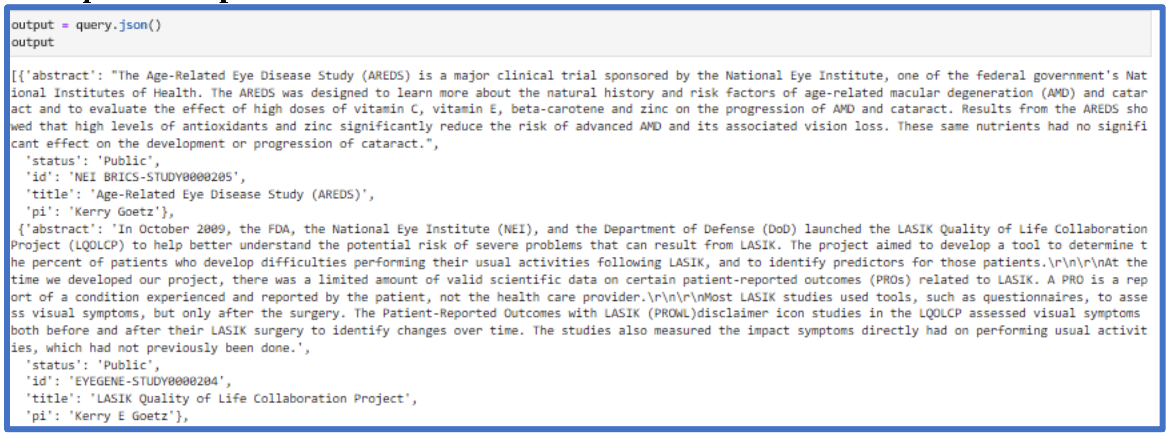
Output:

Output Format	JSON

Output Description:

Output Object	**Required (yes/no)**
Study Title	Yes
Study ID	Yes
Abstract	Yes
Principal Investigator	Yes

Example of Output:



```
output = query.json()
output
```

```
[{'abstract': "The Age-Related Eye Disease Study (AREDS) is a major clinical trial sponsored by the National Eye Institute, one of the federal government's Nat
ional Institutes of Health. The AREDS was designed to learn more about the natural history and risk factors of age-related macular degeneration (AMD) and catar
act and to evaluate the effect of high doses of vitamin C, vitamin E, beta-carotene and zinc on the progression of AMD and cataract. Results from the AREDS sho
wed that high levels of antioxidants and zinc significantly reduce the risk of advanced AMD and its associated vision loss. These same nutrients had no signifi
cant effect on the development or progression of cataract.",
  'status': 'Public',
  'id': 'NEI BRICS-STUDY0000205',
  'title': 'Age-Related Eye Disease Study (AREDS)',
  'pi': 'Kerry Goetz'},
 {'abstract': 'In October 2009, the FDA, the National Eye Institute (NEI), and the Department of Defense (DoD) launched the LASIK Quality of Life Collaboration
Project (LQOLCP) to help better understand the potential risk of severe problems that can result from LASIK. The project aimed to develop a tool to determine t
he percent of patients who develop difficulties performing their usual activities following LASIK, and to identify predictors for those patients.\r\n\r\nAt the
time we developed our project, there was a limited amount of valid scientific data on certain patient-reported outcomes (PROs) related to LASIK. A PRO is a rep
ort of a condition experienced and reported by the patient, not the health care provider.\r\n\r\nMost LASIK studies used tools, such as questionnaires, to asse
ss visual symptoms, but only after the surgery. The Patient-Reported Outcomes with LASIK (PROWL)disclaimer icon studies in the LQOLCP assessed visual symptoms
both before and after their LASIK surgery to identify changes over time. The studies also measured the impact symptoms directly had on performing usual activit
ies, which had not previously been done.',
  'status': 'Public',
  'id': 'EYEGENE-STUDY0000204',
  'title': 'LASIK Quality of Life Collaboration Project',
  'pi': 'Kerry E Goetz'},
```

Getting All Studies Associated with A Form Structure

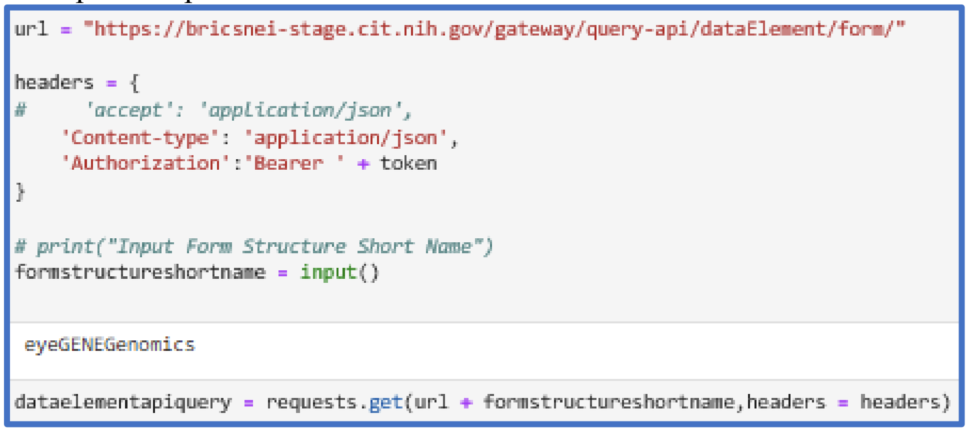
Returns all the studies that have data submitted to the form structure.

Endpoint URL: https://brics.nei.nih.gov/gateway/query-api/study/form?formName=

Method	GET
Headers	Response content type: application/json Content-Type: application/json Authorization: Bearer + Token
Date	Form Structure Short Name

Get Response	`response = requests.get(`
	`"https://https://brics.nei.nih.gov/gateway/queryapi/study/form?formName=Form Structure Shortname",`
	`headers=headers,`
	`)`

Example of Input:



```
url = "https://bricsnei-stage.cit.nih.gov/gateway/query-api/dataElement/form/"

headers = {
#     'accept': 'application/json',
    'Content-type': 'application/json',
    'Authorization':'Bearer ' + token
}

# print("Input Form Structure Short Name")
formstructureshortname = input()
```

```
eyeGENEGenomics
```

```
dataelementapiquery = requests.get(url + formstructureshortname,headers = headers)
```

Output:

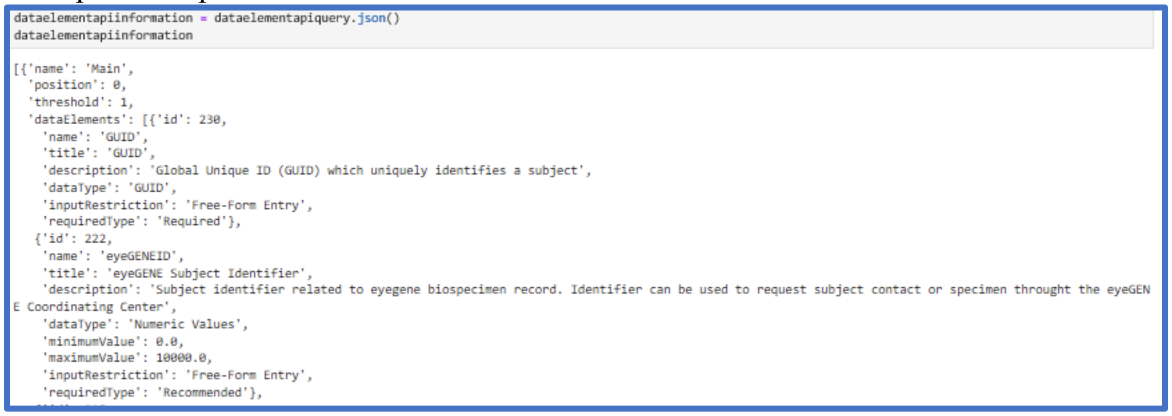
Output Format	JSON

Output Description:

Output Object	**Required (yes/no)**
Data Element Name	Yes
Form Structure Group Name	Yes
Data Element Title	Yes
Data Element Short Description	Yes

Data Element Data Type	Yes

Example of output:



```
dataelementapiinformation = dataelementapiquery.json()
dataelementapiinformation

[{'name': 'Main',
  'position': 0,
  'threshold': 1,
  'dataElements': [{'id': 230,
    'name': 'GUID',
    'title': 'GUID',
    'description': 'Global Unique ID (GUID) which uniquely identifies a subject',
    'dataType': 'GUID',
    'inputRestriction': 'Free-Form Entry',
    'requiredType': 'Required'},
   {'id': 222,
    'name': 'eyeGENEID',
    'title': 'eyeGENE Subject Identifier',
    'description': 'Subject identifier related to eyegene biospecimen record. Identifier can be used to request subject contact or specimen throught the eyeGEN
E Coordinating Center',
    'dataType': 'Numeric Values',
    'minimumValue': 0.0,
    'maximumValue': 10000.0,
    'inputRestriction': 'Free-Form Entry',
    'requiredType': 'Recommended'},
```

Data API

The Data API allows users to retrieve data for one or more studies and form structures. Like the Query Tool, users can:

1. Download data for form structures within a study.
2. Join form structures for one or more studies.
3. Filter on data elements with the advance Boolean Search.

The Data API provides the following endpoints.

HTTPS Method	Path	Description
POST	/data/csv	Retrieve data from a query on a single form or joining up to five forms by GUID. The user can specify multiple studies to collect data. The data will be returned as a zip file containing the query results in a CSV.
POST	/data/csv/files	Retrieve data from a query like in */data/csv* but include any associated files in the returned zip file.
POST	/data/csvChunk	Retrieve query results like */data/csv* but allow the zip file to be downloaded in multiple chunks.
POST	/data/csv/filesChunk	Retrieve the data and files from a query like in */data/csv/files* but allow the user to download the zip file in multiple chunks.
POST	/data/json	Retrieve query results like */data/csv* but instead returns a zip file containing the results in JSON format.
POST	/data/json/files	Retrieve query results like */data/csv/files* but the query results inside the zip file will be in the JSON format.
POST	/data/bulk/study/form	Retrieve query results where the client specifies which forms to retrieve for one or more studies. The response is a zip file with one CSV for each form structure with the combined data across the studies for which that form was specified,
POST	/data/bulk/study/formChunk	Retrieve query results like in */data/bulk/study/form* but allow the user to download the zip file in chunks.
POST	/data/bulk/form/study	Retrieve query results where the data is separated by form structure compiled across one or more studies. The response is a zip

		file containing one CSV for each form structure.
POST	/data/bulk/form/studyChunk	Retrieve query results like in */data/bulk/form/study* but allow the user to download the zip file in chunks.

Each of the "chunk" endpoints works by having the user make the query request along with a byte range in the first request. The response's header will contain a filename and its payload will contain those first requested bytes. On subsequent requests, the user should include this filename as a property of the JSON request body to download the remaining bytes without re-running the query. A specific example is provided in the section below *Download Query Results in Chunks*.

Below is the example endpoint information..

GET DATA FROM MULTIPLE FORM STRUCTURES WITHOUT DOING JOINS (/data/bulk/form/study)

This endpoint returns data for multiple form structures without doing. For each form structure, the client can specify which studies they want the results from. The response will be a zip file containing one CSV for each form structure. These CSVs will have

For example, suppose a user requests to get data from FormX from studies 1 and 2 and data from FormY for studies 2 and 3, and FormZ without any studies specified. The zip file in the response will have three files:

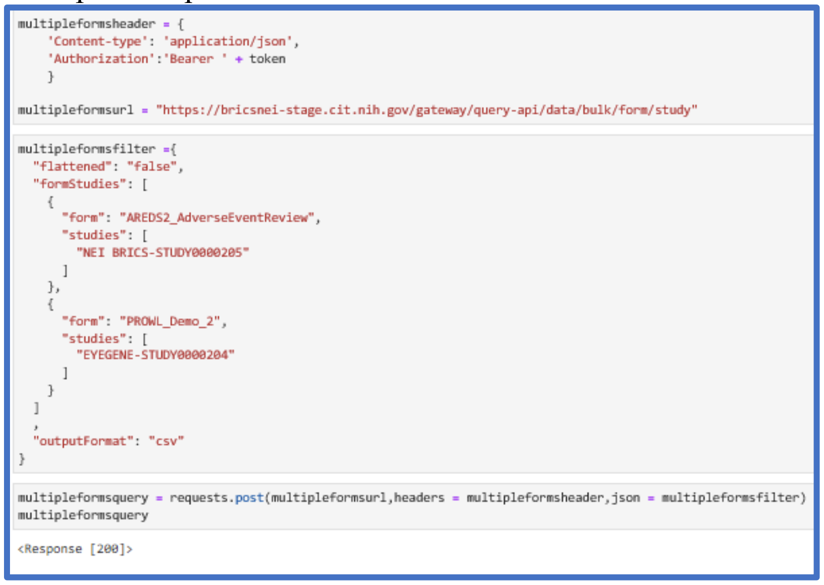
1. query_response_FormX_<TIMESTAMP>,csv
   a. This file will contain data from study 1 and study 2.
2. query_response_FormY_<TIMESTAMP>.csv
   a. This file will contain data from study 2 and study 3.
3. query_response_FormZ_<TIMESTAMP>.csv
   a. This file will contain data from all studies with FormZ data.

Endpoint URL: https://brics.nei.nih.gov/gateway/query-api/data/bulk/form/study

Method	POST
Headers	Content-Type: application/json Authorization: Bearer + Token
Date	Form Structure Short Name Optional: Study Prefix IDs
Response	response = requests.post(

	```
	    "https://https://brics.nei.nih.gov/gateway/queryapi/data/bulk/form/study",
	    headers=headers,
	    json=data,
	)
	```

Example of Input:



```
multipleformsheader = {
    'Content-type': 'application/json',
    'Authorization':'Bearer ' + token
    }

multipleformsurl = "https://bricsnei-stage.cit.nih.gov/gateway/query-api/data/bulk/form/study"

multipleformsfilter ={
  "flattened": "false",
  "formStudies": [
    {
      "form": "AREDS2_AdverseEventReview",
      "studies": [
        "NEI BRICS-STUDY0000205"
      ]
    },
    {
      "form": "PROWL_Demo_2",
      "studies": [
        "EYEGENE-STUDY0000204"
      ]
    }
  ]
  ,
  "outputFormat": "csv"
}

multipleformsquery = requests.post(multipleformsurl,headers = multipleformsheader,json = multipleformsfilter)
multipleformsquery

<Response [200]>
```

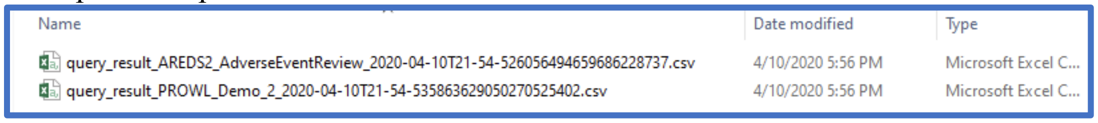
Output:

Output Format	Zip Files

Output Description:

Output Object	**Required (yes/no)**
CSV File of Data	Yes

Example of output:



Name	Date modified	Type
query_result_AREDS2_AdverseEventReview_2020-04-10T21-54-5260564946596862287370.csv	4/10/2020 5:56 PM	Microsoft Excel C...
query_result_PROWL_Demo_2_2020-04-10T21-54-535863629050270525402.csv	4/10/2020 5:56 PM	Microsoft Excel C...

GET DATA FROM MULTIPLE FORM STRUCTURES FOR THE GIVEN STUDIES (/data/bulk/study/form)

This endpoint returns data for multiple studies with optionally specified form structures without performing any joins on the data. If a study does not have a form structure specified, all forms with data for that study will be retrieved.

For example, suppose a user is interested in the following studies which have data for the specified form structures:

1. Study 1
    a. FormX
    b. FormY
2. Study 2
    a. FormX
    b. FormZ
3. Study 3
    a. FormX
    b. FormY
    c. FormZ

Now suppose the user makes the request for the following "studyForms" data:

1. Study 1
    a. FormX
2. Study 2
    a. FormX
    b. FormZ
3. Study 3

The zip file in the response will contain the following files:

1. query_response_FormX_<TIMESTAMP>,csv
    a. This file will contain data from study 1, study 2, and study 3.
2. query_response_FormY_<TIMESTAMP>.csv
    a. This file will contain data from study 3.
3. query_response_FormZ_<TIMESTAMP>.csv
    a. This file will contain data from study 2 and study 3.

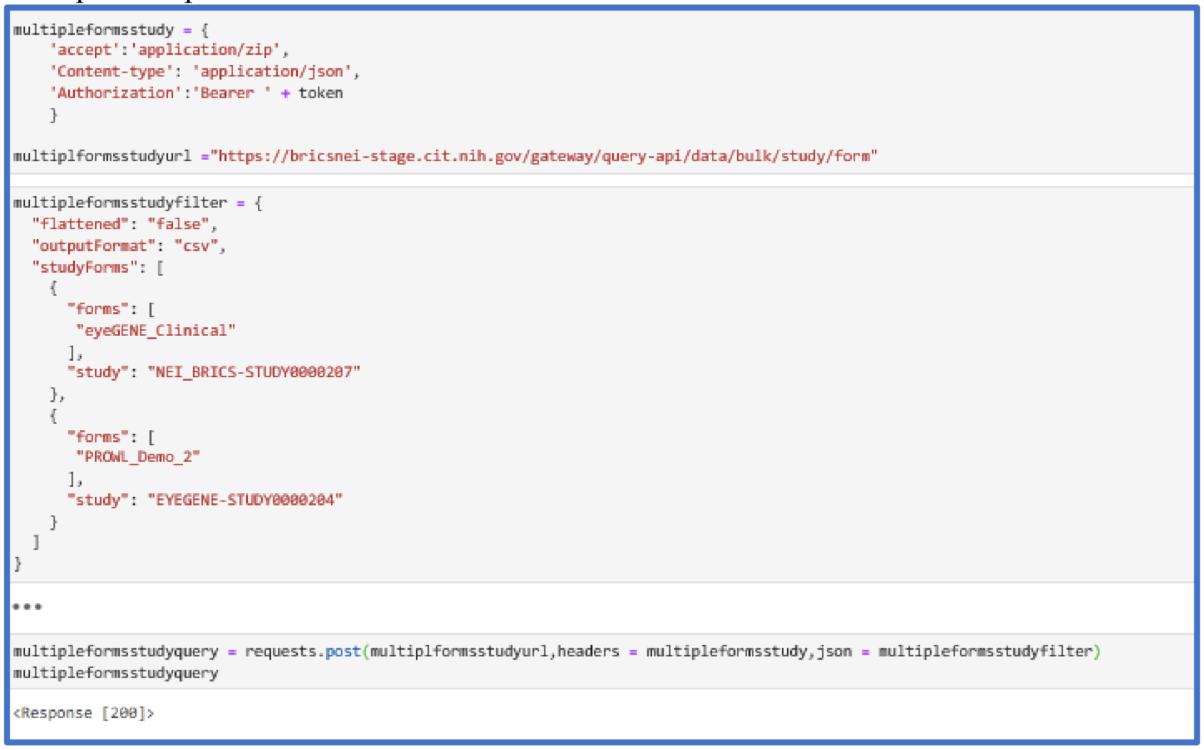
Again, note that since no forms were specified for study 3, data for all of the form structures are retrieved.

Endpoint URL: https://brics.nei.nih.gov/gateway/query-api/data/bulk/study/form

Method	POST
Headers	Response Content Type: application/zip Content-Type: application/json Authorization: Bearer + Token
Date	Form Structure Short Name Optional: Study Prefix IDs

Response	`response = requests.post(`
	`"https://https://brics.nei.nih.gov/gateway/queryapi/data/bulk/study/form",`
	`headers=headers,`
	`json=data,`
	`)`

Example of Input:



```
multipleformsstudy = {
    'accept':'application/zip',
    'Content-type': 'application/json',
    'Authorization':'Bearer ' + token
    }

multiplformsstudyurl ="https://bricsnei-stage.cit.nih.gov/gateway/query-api/data/bulk/study/form"
```

```
multipleformsstudyfilter = {
  "flattened": "false",
  "outputFormat": "csv",
  "studyForms": [
    {
      "forms": [
       "eyeGENE_Clinical"
      ],
      "study": "NEI_BRICS-STUDY0000207"
    },
    {
      "forms": [
       "PROWL_Demo_2"
      ],
      "study": "EYEGENE-STUDY0000204"
    }
  ]
}
```

```
•••
```

```
multipleformsstudyquery = requests.post(multiplformsstudyurl,headers = multipleformsstudy,json = multipleformsstudyfilter)
multipleformsstudyquery
```

```
<Response [200]>
```

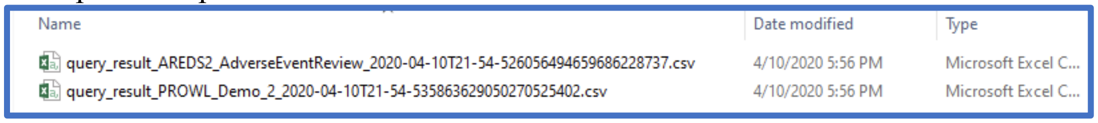
Output:

Output Format	Zip Files

Output Description:

Output Object	**Required (yes/no)**
CSV File of Data	Yes

Example of output:



```
Name                                                                  Date modified        Type
query_result_AREDS2_AdverseEventReview_2020-04-10T21-54-526056494659686228737.csv   4/10/2020 5:56 PM   Microsoft Excel C...
query_result_PROWL_Demo_2_2020-04-10T21-54-535863629050270525402.csv                4/10/2020 5:56 PM   Microsoft Excel C...
```

Retrieve DATA WITH FILTER AND JOINS

This endpoint returns data based on the specified query. This query can include data element value filters as well as joins between up to five form structures where the forms are joined on GUID. Data can be returned in two formats:

1. CSV
2. JSON

Note that the datatype of the output is specified by the endpoint URL rather than by modifying the "Accept" header value in the request. Additionally, the JSON version of the endpoint is not available for the similar endpoints that also return associated files. The endpoints for these two different response data formats are listed below.

1. CSV endpoint - https://brics.nei.nih.gov/gateway/query-api/data/csv
2. JSON endpoint - https://brics.nei.nih.gov/gateway/query-api/data/json

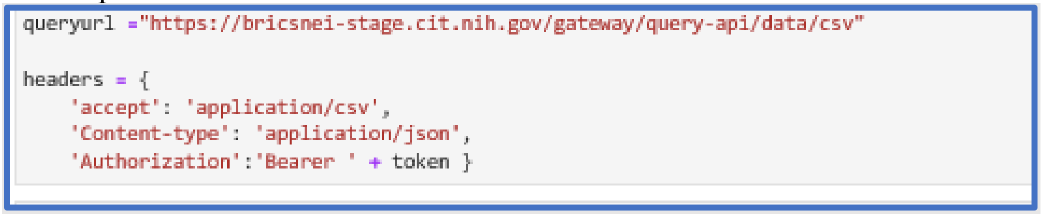
The *Filter* schema is fully specified in the Swagger documentation linked above. It should be noted that the filter property of the JSON body must be an array of *Filter* objects. This array can be empty if no filters are to be used.

Method	POST
Headers	Response Content Type: application/csv or application/json Content-Type: application/json Authorization: Bearer + Token
Data	Form Structure Short Name Optional: Study Prefix IDs

Response	`response = requests.post(`
	`"https://https://brics.nei.nih.gov/gateway/queryapi/data/csv",`
	`headers=headers,`
	`json=data,`
	`)`

Example of Input:

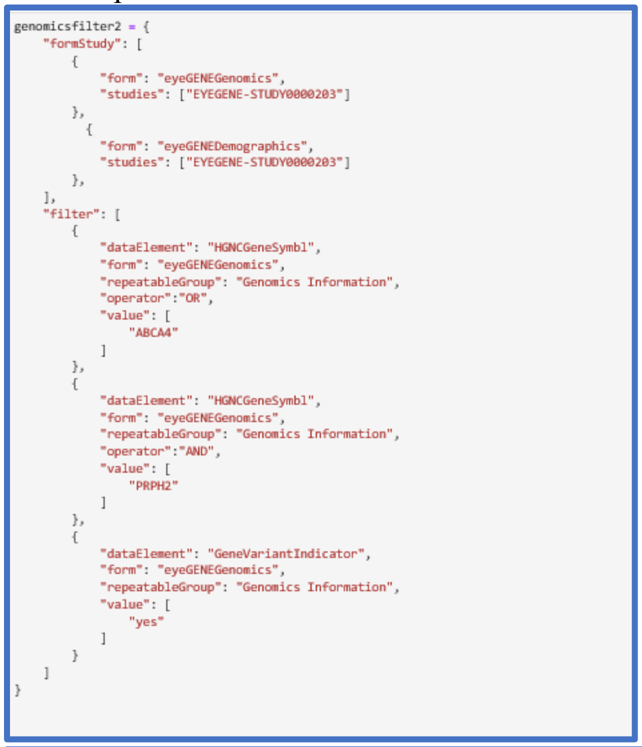
CSV Input:



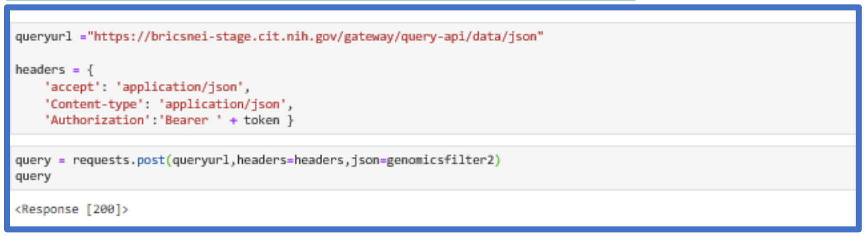
```
queryurl ="https://bricsnei-stage.cit.nih.gov/gateway/query-api/data/csv"

headers = {
    'accept': 'application/csv',
    'Content-type': 'application/json',
    'Authorization':'Bearer ' + token }
```

JSON Input:



```
genomicsfilter2 = {
    "formStudy": [
        {
            "form": "eyeGENEGenomics",
            "studies": ["EYEGENE-STUDY0000203"]
        },
          {
            "form": "eyeGENEDemographics",
            "studies": ["EYEGENE-STUDY0000203"]
        },
    ],
    "filter": [
        {
            "dataElement": "HGNCGeneSymbl",
            "form": "eyeGENEGenomics",
            "repeatableGroup": "Genomics Information",
            "operator":"OR",
            "value": [
                "ABCA4"
            ]
        },
        {
            "dataElement": "HGNCGeneSymbl",
            "form": "eyeGENEGenomics",
            "repeatableGroup": "Genomics Information",
            "operator":"AND",
            "value": [
                "PRPH2"
            ]
        },
        {
            "dataElement": "GeneVariantIndicator",
            "form": "eyeGENEGenomics",
            "repeatableGroup": "Genomics Information",
            "value": [
                "yes"
            ]
        }
    ]
}
```



```
queryurl ="https://bricsnei-stage.cit.nih.gov/gateway/query-api/data/json"

headers = {
    'accept': 'application/json',
    'Content-type': 'application/json',
    'Authorization':'Bearer ' + token }
```

```
query = requests.post(queryurl,headers=headers,json=genomicsfilter2)
query
```

```
<Response [200]>
```

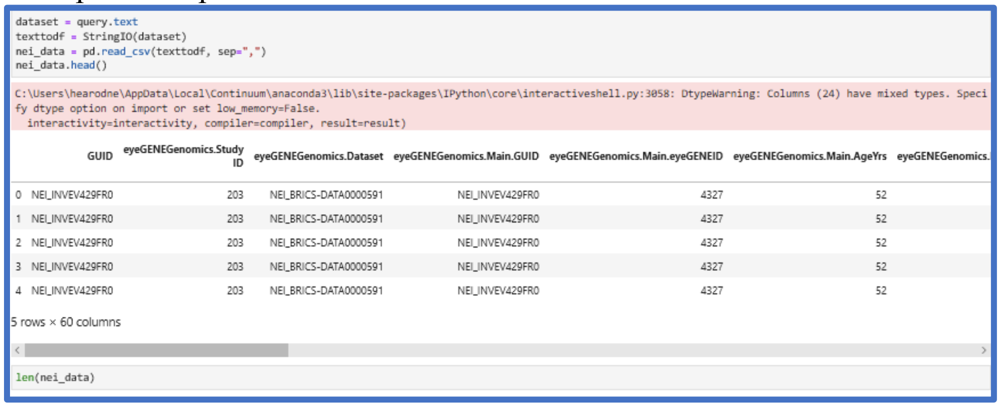
Output for CSV:

Output Format	Text

Output Description:

Output Object	**Required (yes/no)**
CSV File of Data	Yes

Example of output for CSV:



```
dataset = query.text
texttodf = StringIO(dataset)
nei_data = pd.read_csv(texttodf, sep=",")
nei_data.head()
```

```
C:\Users\hearodne\AppData\Local\Continuum\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3058: DtypeWarning: Columns (24) have mixed types. Speci
fy dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
              GUID  eyeGENEGenomics.Study  eyeGENEGenomics.Dataset  eyeGENEGenomics.Main.GUID  eyeGENEGenomics.Main.eyeGENEID  eyeGENEGenomics.Main.AgeYrs  eyeGENEGenomics.
                                       ID
0  NEI_INVEV429FR0                    203  NEI_BRICS-DATA0000591          NEI_INVEV429FR0                            4327                           52
1  NEI_INVEV429FR0                    203  NEI_BRICS-DATA0000591          NEI_INVEV429FR0                            4327                           52
2  NEI_INVEV429FR0                    203  NEI_BRICS-DATA0000591          NEI_INVEV429FR0                            4327                           52
3  NEI_INVEV429FR0                    203  NEI_BRICS-DATA0000591          NEI_INVEV429FR0                            4327                           52
4  NEI_INVEV429FR0                    203  NEI_BRICS-DATA0000591          NEI_INVEV429FR0                            4327                           52

5 rows × 60 columns
```

```
len(nei_data)
```

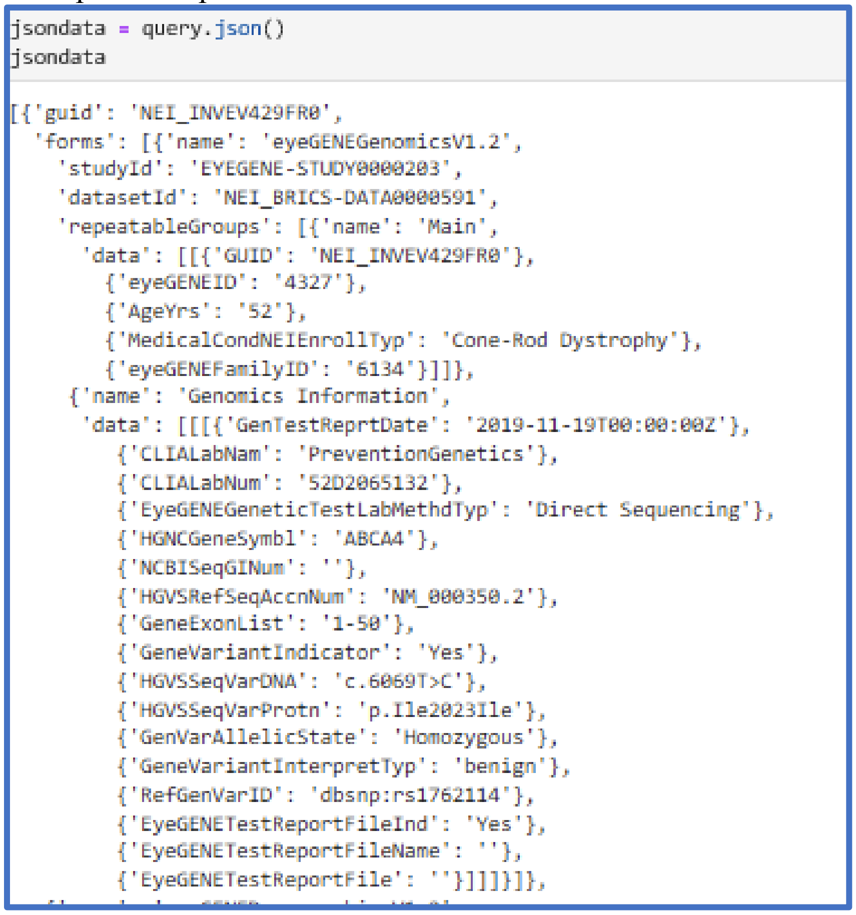
Output for CSV:

Output Format	JSON

Output Description:

Output Object	**Required (yes/no)**
GUID	Yes
Form Structure Short Name	Yes
Study ID	Yes
Dataset ID	Yes
Form Structure Repeatable Group	Yes
Data Elements and Associated Data	Yes

Example of output for JSON:



```
jsondata = query.json()
jsondata
```

```
[{'guid': 'NEI_INVEV429FR0',
  'forms': [{'name': 'eyeGENEGenomicsV1.2',
    'studyId': 'EYEGENE-STUDY0000203',
    'datasetId': 'NEI_BRICS-DATA0000591',
    'repeatableGroups': [{'name': 'Main',
      'data': [[{'GUID': 'NEI_INVEV429FR0'},
        {'eyeGENEID': '4327'},
        {'AgeYrs': '52'},
        {'MedicalCondNEIEnrollTyp': 'Cone-Rod Dystrophy'},
        {'eyeGENEFamilyID': '6134'}]]},
     {'name': 'Genomics Information',
      'data': [[[{'GenTestReprtDate': '2019-11-19T00:00:00Z'},
         {'CLIALabNam': 'PreventionGenetics'},
         {'CLIALabNum': '52D2065132'},
         {'EyeGENEGeneticTestLabMethdTyp': 'Direct Sequencing'},
         {'HGNCGeneSymbl': 'ABCA4'},
         {'NCBISeqGINum': ''},
         {'HGVSRefSeqAccnNum': 'NM_000350.2'},
         {'GeneExonList': '1-50'},
         {'GeneVariantIndicator': 'Yes'},
         {'HGVSSeqVarDNA': 'c.6069T>C'},
         {'HGVSSeqVarProtn': 'p.Ile2023Ile'},
         {'GenVarAllelicState': 'Homozygous'},
         {'GeneVariantInterpretTyp': 'benign'},
         {'RefGenVarID': 'dbsnp:rs1762114'},
         {'EyeGENETestReportFileInd': 'Yes'},
         {'EyeGENETestReportFileName': ''},
         {'EyeGENETestReportFile': ''}]]]}]},
```

Download the Query Data with Associated Files in Chunks (/data/csv/filesChunk)

This endpoint allows the user to download the query response and all files associated with those records in chunks (i.e. via multiple HTTPS requests that each retrieve a portion of the final zip file).

The base API endpoints work on the principle of returning all specified data in a single request. However, when downloading imaging or genomics data, the size of the data returned is often too large to be reasonably contained in a single response. The chunked file endpoints provide users with a way to download these large results through multiple HTTPS requests without having to re-run the underlying query.

To use a chunked file endpoint, the user will first make the initial request with the same request body as before. However, the user should specify the request "Range" header to only retrieve the first chunk of the file. For example, to retrieve the first 10 MB of the file, the user would set the "Range" header to have value "0-10485760". The response body will then contain the specified content range. The response headers will contain information about the total number of bytes in the "Content-Range" header. The header "Content-Disposition" contains the name of the file. The user will need to include this file name in the JSON body under property "fileName" to avoid re-running the query. To get the remainder of the file, the user will make the same request with the "fileName" property included and the "Range" header incremented until the entire file has been downloaded.

Since this is a complicated process, here is an example case in more detail. Suppose a user is interested in Study 1 which has 5 GB of ImagingMR data on FITBIR. The user will make the following POST request to make the query and retrieve the first 10 MB of the resulting zip file:

```
curl -X POST -H 'Authorization: Bearer <TOKEN>' -H 'Content-Type: application/json' -H
'Range: bytes=0-10485760' -H 'accept: application/octet-stream' -d '{"formStudy":
[{"form": "ImagingMR", "studies": ["FITBIR-STUDY0000001"]}], "filter": [],
"flattened": false}' https://fitbir.nih.gov/gateway/query-api/data/csv/filesChunk -o
myApiExample.zip
```

The response body will include the first 10 MB of the zip file. It will also include the following headers:

1. `Content-Range: bytes 0-10485760/5368709120`
   a. This specifies that the returned bytes were from byte 0 to byte 10485760.
   b. It also states that the total size of the file is 5368709120 bytes (5 GB).
2. `Content-disposition: attachment; filename=serverExampleAPIFile.zip`
   a. This states that the file name is "serverExampleAPIFile.zip".

b. *Note: This header is typically written with the "D" in disposition capitalized but these endpoints may return this header with that character in lower-case. HTTPS headers are intended to be case-insensitive, so this is only an issue if the headers are being manually parsed by the user's program.*

Now the user can get the next 10 MB chunk through the following POST request:

```
curl -X POST -H 'Authorization: Bearer <TOKEN>' -H 'Content-Type: application/json' -H
'Range: bytes=10485761-20971520' -H 'accept: application/octet-stream' -d
'{"formStudy": [{"form": "ImagingMR", "studies": ["FITBIR-STUDY0000001"]}], "filter":
[], "flattened": false, "fileName": "queryApi-2025-07-18T12-42-27.zip"}'
https://fitbir.nih.gov/gateway/query-api/data/csv/filesChunk >> myApiExample.zip
```

Note the following changes:

1. The "Range" header has been updated to retrieve the next 10 MB.
2. The JSON body has been updated to include the file name that was included in the response headers.
3. The content results are being appended to the file on the client-side (by using ">>") rather than overwriting the first 10 MB.

To download the entire 5 GB file, the user would need to write a script to repeat this process 48 more times, updating the "Range" header as appropriate. The resulting zip file will contain a CSV containing the query results along with all of the files associated with those records. In this case, the zip file will contain the 5 GB of imaging data.

Method	POST
Headers	Response Content Type: application/csv Content-Type: application/json Authorization: Bearer + Token
Data	Form Structure Short Name Optional: Study Prefix IDs

Response	`response = requests.post(`
	`"https://https://brics.nei.nih.gov/gateway/queryapi/data/bulk/form/study",`
	`headers=headers,`
	`json=data,`
	`)`

Appendix

Attached are sample API scripts:

Script Name	Description
BRICSAPI_UserGuide	Provides examples of the API Endpoints
API_Case1	From the join of the two form structures, eyeGeneDemographics and eyeGeneGenomics, provide the GUIDs with one gene and one or more gene variant types.
API_Case2	The purpose of the script is to provide the list of GUIDs from the eyeGeneGenomics form structure that have one gene and one or more gene variant types
API_Case3A	The purpose of the script provides the GUIDs for a gene but does not include a gene variant type.
API_Case3B	The purpose of the script provides the GUIDs with multiple genes and gene variant type.
API_GenesandExcludeGeneVariant	The purpose of the script provides the list of GUIDs with two genes and a gene variant type but excludes additional gene variant type.
API_Genevarianttype	The purpose of the script is to provide the list of GUIDSs with two genes and a gene variant interpretation type
API_GUIDsnotinGenomics	The purpose of this script is to return the GUIDs that exist in the eyeGeneDemographics data, but not in the eyeGeneGenomics data.
HGNCGeneSymblandGeneIndicator	The purpose of this script is to return all GUIDs that have two genes and the gene indicator is “yes”.