

DH01

ClinicalTrials.gov REST API – tapping into the clinical studies registry database the smart way

Lukasz Kniola, Biogen, Maidenhead, UK

ABSTRACT

ClinicalTrials.gov is a treasure trove of information about clinical studies across the world, provided and updated by the study sponsors. Most users know it as a website where the registry database can be manually searched and queried, and results viewed or downloaded.

But there is another way of tapping into this resource, which can be incorporated into programs, scripts, and workflows. ClinicalTrials.gov offers a free, flexible, and configurable REST API (application programming interface) which duplicates and extends the capabilities of the website and allows to query the database and import the results as JSON, CSV or XML files for further processing and analysing, or to support local tools.

This paper will introduce the common methods and parameters available in the API, explain the value of having this resource “on-tap”, and offer ideas of using it to automate workflows not just in the clinical trial transparency domain.

INTRODUCTION

Each clinical trial sponsor holds their own database of historical, ongoing, and upcoming clinical trials for their internal purposes and benefits. In parallel, there exist industry registries where many details of clinical studies from multiple sponsors are publicly available, and the information can be searched, viewed, and downloaded. Different regulators require studies run in their jurisdictions to be recorded in their relevant registries. Most notably, studies run in the United States and the European Union must be registered in ClinicalTrials.gov and EudraCT/EU CTR respectively.

Due to the global nature of our industry where many studies are being run across multiple geographies and submitted for approval to multiple agencies, there is a cross-population of the records in different registries.

This information can be used by the public to learn about clinical trials and therapies, but it can equally be a great resource for sponsors to view individual and aggregate information about their own studies as well as the wider industry.

WHAT IS CLINICALTRIALS.GOV AND WHAT INFORMATION CAN I FIND THERE?

ClinicalTrials.gov is a website and a database which serves patients, physicians, researchers, and the public as a tool to find and view information on clinical studies past and present conducted in the United States, but also other countries (219 in total). The information is kept up to date by the study sponsors. Ideally, studies are added to the database as soon as they start, and the records are updated as the studies continue.

Each record in ClinicalTrials.gov contains information about the study protocol. The details include type of disease and intervention, study description, study design, inclusion/exclusion criteria, countries, treatment arms, endpoints, milestone dates, status, and many more. Often, summary information on study participants (number of participants starting and completing the study) are also present. On some occasions, study results are added upon completion. This may include further description of study participants (demographic data, etc.), outcomes of the study, summary of adverse events.

All this information can be queried and listed using the website’s search tool. The results can be viewed and downloaded in different formats for further processing.

WHAT IS A REST API AND WHY USE IT?

A REST API (also known as RESTful API) is an application programming interface (API or web API) that allows to interact with web services.

REST stands for Representational State Transfer. In technical terms this means that when a client sends a request to query a resource served by the REST API, the server responds by transferring back the current state of the resource in a standardized representation. Put simply, it enables sending a query over the web in the form of a URL using pre-defined parameters (and in some cases simple syntax) and receiving relevant information in a format that can be easily interpreted and used for further processing.

Apart from viewing, REST APIs can also be used to modify items (records) on the server, add new or remove existing items, although this is not in scope of this paper. Most modern websites and services have APIs available to developers. This allows to build new tools and capabilities which take advantage of existing resources.

ClinicalTrials.gov API is an open API which does not require authentication or registering accounts in order to use it. It has a generous limit of requests that can be sent in a short span of time and generally few limitations on use and capabilities.

It is simple enough to use the ClinicalTrials.gov website search tool to find relevant information, download the results and feed them into a separate app or program to analyse them further. However, this becomes impractical if the query needs to be sent on a regular basis or is a step in a program or a macro. Instead of performing multiple manual steps, querying and retrieving results can be “baked” into programs, which can then immediately proceed to processing and visualising the relevant information.

COMMON METHODS AND PARAMETERS IN CLINICALTRIALS API

ClinicalTrials API is a versatile and flexible tool, with many options to access any and all information posted to the website by sponsors. It allows for sending simple general queries as well as quite complex multi-parameter expressions to narrow down the search criteria.

The full documentation, describing all capabilities and options is available on the API website:

<https://clinicaltrials.gov/api/gui/home>, https://clinicaltrials.gov/api/gui/ref/api_urls.

This paper introduces only a subset of options, to focus on practical applications and offer a taste of the possibilities which can lead to the reader's further exploration of what's available and helpful for their own needs.

THE BASIC STRUCTURE OF A QUERY URL

In simple terms, the query can be sent by typing an URL in a web browser or submitting a HTTP request programmatically. The structure of the URL will follow the same pattern:

`https://ClinicalTrials.gov/api/query/ TypeOfRequest ? param=value1 & param=value2`

COMMON PARAMETERS USED IN THE QUERIES

expr= search expression (more details later)

min_rnk= the first record returned (e.g. first record found can be omitted by specifying min_rnk=2, although in practical terms this is either left empty or set to 1)

max_rnk= the last record returned (the query will return max_rnk – min_rnk + 1 results)

fmt= the format in which the results are returned (depending on the type of request, this can be XML, JSON, CSV or TREE).

If min_rnk/max_rnk parameters are not specified, each request type will return its default number of records.

TYPES OF REQUESTS

The most used and the most versatile type of request is **Study_Fields**. It allows to query the registry for specific expressions and fields, narrowing down the information returned to the specific needs of the user. The next section of the paper will focus on this request type.

- **Study Fields** (“study_fields”) – finds and returns values of specific fields for a large set of study records. The full name of available fields can be found here: <https://clinicaltrials.gov/api/gui/ref/crosswalks> or https://clinicaltrials.gov/api/info/study_fields_list. By default, this request returns 20 study records. This can be altered by specifying min_rnk/max_rnk parameters.

Example query:

`https://clinicaltrials.gov/api/query/study_fields?expr=covid&fields=NCTId,LeadSponsorName,BriefTitle,Condition&min_rnk=1&max_rnk=5&fmt=csv`

Result:

"APIVrs: 1.01.03"

"DataVrs: 2021:09:27 22:32:33.007"

"Expression: covid"

"NStudiesAvail: 390761"

"NStudiesFound: 7272"

"MinRank: 1"

"MaxRank: 5"

"NStudiesReturned: 5"

"Field Names: NCTId,LeadSponsorName,BriefTitle,Condition"

"Rank","NCTId","LeadSponsorName","BriefTitle","Condition"

1,"NCT04609969","Centre Hospitalier Régional d'Orléans","Diagnostic Performance Evaluation of a Novel SARS-CoV-2 (COVID-19) Antigen Detection Test","Covid19|SARS-CoV-2 Infection"

2,"NCT04785898","Groupe Hospitalier Paris Saint Joseph","Diagnostic Performance of the ID Now™ COVID-19 Screening Test Versus Simplexa™ COVID-19 Direct Assay","Covid19"

3,"NCT04412265","University of Milano Bicocca","Frailty in Elderly Patients With COVID-19","Covid19"

```
4,"NCT04476940","Meharry Medical College","COVID-19 Breastfeeding Guideline for African-  
Americans","Covid19|Exclusive Breastfeeding"  
5,"NCT04427345","University of Milano Bicocca","Predictive Factors COVID-19  
Patients","covid19"
```

There are two more main types of requests:

- **Full Studies** ("full_studies") – finds and returns all details held on a study. By default, it returns the first study record which meets the criteria specified in `expr=` parameter. This can be changed by specifying minimum and maximum rank parameters (`min_rnk=`, `max_rnk=`), although this type of request should not be used to retrieve information from many studies.

Example query (copy the url below to see full results in your browser):

https://clinicaltrials.gov/api/query/full_studies?expr=covid-19&min_rnk=1&max_rnk=20&fmt=json

- **Field Values** ("field_values") - Returns all unique values from a particular field for a set of study records. Select the field values returned by using the field parameter (shown in the Query Parameters table). The full name of available fields can be found here: <https://clinicaltrials.gov/api/gui/ref/crosswalks> or https://clinicaltrials.gov/api/info/study_fields_list.

DATA ELEMENTS AND FIELD NAMES

There is a vast array of details contained in the ClinicalTrials database. As of API version 1.01.03, there are 322 fields that are viewable and searchable. The **registration elements** include information on:

- study identification (protocol number, title, IDs, descriptions, etc.),
- study status (recruitment status, start and completion dates),
- sponsor (details of sponsor, collaborators, investigators),
- FDA oversight info,
- study description and keywords,
- study design (purpose, phase, enrolment count),
- arms, groups, and interventions (type, title, description),
- outcome measures (primary, secondary, and other outcome descriptions and time frames),
- eligibility (age limits, inclusion/exclusion criteria, etc.),
- IPD sharing statement,
- and others.

Where reported, **result elements** provide such details as:

- participant flow,
- baseline characteristics,
- outcome measures,
- statistical analyses,
- AE information.

Finally, **key record date** fields include information about when results were posted, submitted, updated, etc.

A comprehensive list and a matchup of data fields found on the website and data fields available in the API can be found here:

<https://clinicaltrials.gov/api/gui/ref/crosswalks>

Some of the fields used in examples and illustrations in this paper include:

NCTId, BriefTitle, OfficialTitle, Acronym, SecondaryId, LeadSponsorName, Phase (Study Phase), StartDate (Study Start Date), ResultsFirstPostDate

but of course, different combination of fields are allowed and can be used for all types of summaries, lists and reports.

THE SEARCH EXPRESSION (EXPR= PARAMETER)

The expression contains the text that is used to search the ClinicalTrials database. This can be a single word (e.g., NCT05058729, Biogen, Poland) or a sequence of words (e.g., "Multiple Sclerosis" – note the speech marks to treat this as a phrase). It is also possible to use Boolean operators – AND, OR, NOT. Those can be in turn grouped in brackets to specify precedence of operators. E.g.:

`expr=Covid AND (USA OR France) AND (AstraZeneca AND NOT (Pfizer OR Moderna))`

NARROWING DOWN THE SEARCH CRITERIA

Although the API applies a grading algorithm to different fields (e.g., text found in the study title is ranked higher than the same text found in investigator detail), the examples above still search all fields to generate the results of the query. In certain scenarios it may be necessary to narrow down the search to make sure that only a specific field is checked.

To achieve this, another operator can be used – AREA. This specifies the field to be searched. E.g., to find all records listing Biogen as a lead sponsor, the following expression can be used:

```
expr=AREA[LeadSponsorName]Biogen
```

Note that the same Boolean operators can be used in combination with AREA. E.g., to list all records which mention Biogen in any field for which Biogen is not the lead sponsor, the following expression can be used:

```
expr=Biogen AND NOT AREA[LeadSponsorName]Biogen
```

The AREA operator may also prove useful when searching for a specific study by its NCT id. E.g., using a simple expression “NCT00390221” returns two records: NCT00390221 (the study we’re after), and NCT00870740 (a study which was an extension to NCT00390221). To limit the search only to the NCT id field, it is advisable to use explicit expression:

```
expr=AREA[NCTId]NCT00390221
```

Note that the field names specified in the AREA operator are case sensitive (requesting nctid instead of NCTId will return an error).

OTHER USEFUL OPERATORS

RANGE is an operator especially useful when selecting dates but can also apply to numeric and alphanumeric values. It is passed on with two parameters defining the min and max values. E.g., the following expression only selects records for studies whose results were first posted on or after 1 Jan 2015:

```
expr=AREA[ResultsFirstPostDate]RANGE[01/01/2015, MAX]
```

COVERAGE can help specify the degree to which the term should match the value in the field: FullMatch, StartsWith, EndsWith, Contains (default). E.g.,

```
expr=AREA[Condition]COVERAGE[FullMatch](pain OR asthma)
```

EXPANSION defines if the term should be taken as is or if it can be expanded to synonyms and concepts: None, Term, Concept, Relaxation (default), Lossy.

MISSING operator is used to find records which have no value in a particular field. E.g.:

```
expr=AREA[ResultsFirstPostDate]MISSING
```

NUANCES OF URL PARSING

When typing or pasting URLs into the web browser, it is perfectly acceptable to use all characters, like comma, brackets, spaces, etc. However, in some instances, URLs containing those characters may cause issues or even be disallowed. To avoid problems, non-alphanumeric characters can be replaced using a hexadecimal notation. Below are some common examples, but other troublesome characters can be swapped using an ASCII table, too.

Standard notation	Replacement for URL
“ ” (space)	%20 or “+”
, (comma)	%2C
‘ (single speech mark)	%27
“ (double speech mark)	%22
[] square brackets	%5B %5D
() round brackets	%28 %29

For example, the expression portion of the URL:

```
expr=AREA[Condition]pain OR asthma
```

May need to be changed to:

```
expr=AREA%5BCondition%5D%28pain%29OR%28asthma%29
```

THE VALUE OF HAVING CLINICAL TRIAL REGISTRY “ON-TAP”

To quickly query the ClinicalTrials database it may be easiest to open the ClinicalTrials.gov website and use the online interface. However, when this exercise needs to be repeated a number of times, or on a regular basis, or the query result is to be fed into a program to process it, it will almost always be easier to take advantage of the API and bake the request and the retrieval of results into the program itself.

The following are three relatively simple examples in three most popular programming languages among statistical programmers, which will hopefully illustrate the ease with which ClinicalTrials database can be looked up. The examples are simple but will hopefully trigger thoughts and ideas of more complex systems where the knowledge stored in the ClinicalTrials database can be combined with the processing tools of choice.

EXAMPLE 1: WHAT IS THAT STUDY'S PROTOCOL NUMBER?

Clinical trials are often referred to using the nicknames (e.g., INSPIRE) and other times using the protocol numbers (e.g., "KX101-2018-30"). Study teams working on specific trials or programs will know by heart all the study nicknames, protocol numbers and other IDs. But teams working across the portfolio may deal with requests which list only the NCT number, or nickname, but may choose to log them using the protocol number.

The following R program sends a request to ClinicalTrials API for a JSON file containing all Biogen study records:

```
CTgov_API.R

library(httr)
library(jsonlite)

# Send and retrieve HTTP (REST) request
res <- GET("https://clinicaltrials.gov/api/query/study_fields?expr=AREA[LeadSponsorName]Biogen&fields=NCTId,OrgStudyId,Acronym,SecondaryId&min_rnk=1&max_rnk=1000&fmt=json")

# Extract contents of request, store as list
data <- fromJSON(rawToChar(res$content))

# Drill to StudyFields and store as Dataframe
studyFields <- as.data.frame(data$StudyFieldsResponse$StudyFields)

# Optionally, remove records where Acronym is empty
acronyms <- studyFields[which(studyFields$Acronym != quote(character(0))),]

View(acronyms)
```

The result is converted into a dataframe with all study IDs, which can then be further processed in R:

CTgov_API.R		acronyms			
	Rank	NCTId	OrgStudyId	Acronym	SecondaryId
1	1	NCT05067790	232SM303	ASCEND	2021-001294-23
2	2	NCT05058950	285PI401	Intuition	
7	7	NCT04961567	230LE304	TOPAZ-2	2020-005776-35
8	8	NCT04948606	CA-VUM-11892	EXPER-CA/IL	
9	9	NCT04924153	992EP001	K1Te	
11	11	NCT04895241	230LE303	TOPAZ-1	2020-005775-12
12	12	NCT04856982	233AS303	ATLAS	2020-004590-51
13	13	NCT04832399	PRT-TYS-12-10409	TYPIFI	
15	15	NCT04756700	FR-MSG-11654	DigiToms	2020-A01801-38
16	16	NCT04756687	FR-BGT-11758	Lympho-TEC	
19	19	NCT04729907	232SM302	ONWARD	2020-004708-32

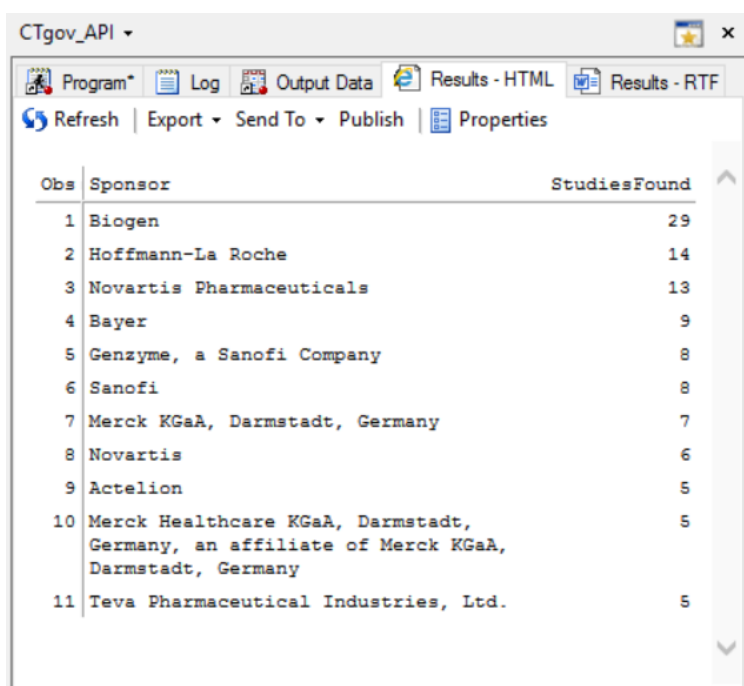
EXAMPLE 2: SPONSORS RUNNING STUDIES FOR CERTAIN CONDITION IN A PARTICULAR COUNTRY

The following example is a SAS® program with a request to ClinicalTrials API to retrieve information on Multiple Sclerosis studies run in Poland. The program uses the “fields_values” query which returns all *LeadSponsorName* field values along with numbers of records related to the search expression and sponsor. The results are then sorted in descending order and limited to sponsors with 5 or more studies.

```
CTgov_API.sas
filename request HTTP
'https://clinicaltrials.gov/api/query/field_values?expr=AREA[LocationCountry] Poland%
20AND%20AREA[Condition]Multiple%20Sclerosis&field=LeadSponsorName&fmt=csv' debug;

proc import datafile=request
            out=ct(rename=(VAR1=n VAR2=ValuesFound VAR3=StudiesFound VAR4=Sponsor))
            dbms=csv replace;
    getnames=no;
    datarow=14;
run;

proc sort data=ct;
    by descending StudiesFound;
    where StudiesFound ge 5;
run;
```



Obs	Sponsor	StudiesFound
1	Biogen	29
2	Hoffmann-La Roche	14
3	Novartis Pharmaceuticals	13
4	Bayer	9
5	Genzyme, a Sanofi Company	8
6	Sanofi	8
7	Merck KGaA, Darmstadt, Germany	7
8	Novartis	6
9	Actelion	5
10	Merck Healthcare KGaA, Darmstadt, Germany, an affiliate of Merck KGaA, Darmstadt, Germany	5
11	Teva Pharmaceutical Industries, Ltd.	5

Note that some Sponsor values are reported with slightly different wording (see obs 3 and 8, as well as 7 and 10), so the program would benefit from additional data cleaning and post-processing to make sure all records belonging to the same category are summed up correctly.

This examples illustrates how with only two proc statements it is possible to produce a meaningful summary, which can very easily be scaled up to include multiple geographies, conditions, and other parameters.

EXAMPLE 3: RECRUITMENT AND STUDY COUNT FOR A SPECIFIC CONDITION OVER YEARS

ClinicalTrials.gov can be queried for any records of studies on a specific condition. The following example Python program requests a CSV file with all records related to "Multiple Sclerosis". Then, using simple Pandas manipulation and the Streamlit package it is possible to create a report of visualize study count and enrolment over time for this particular condition. It allows to select/deselect phases which should be included in the report.

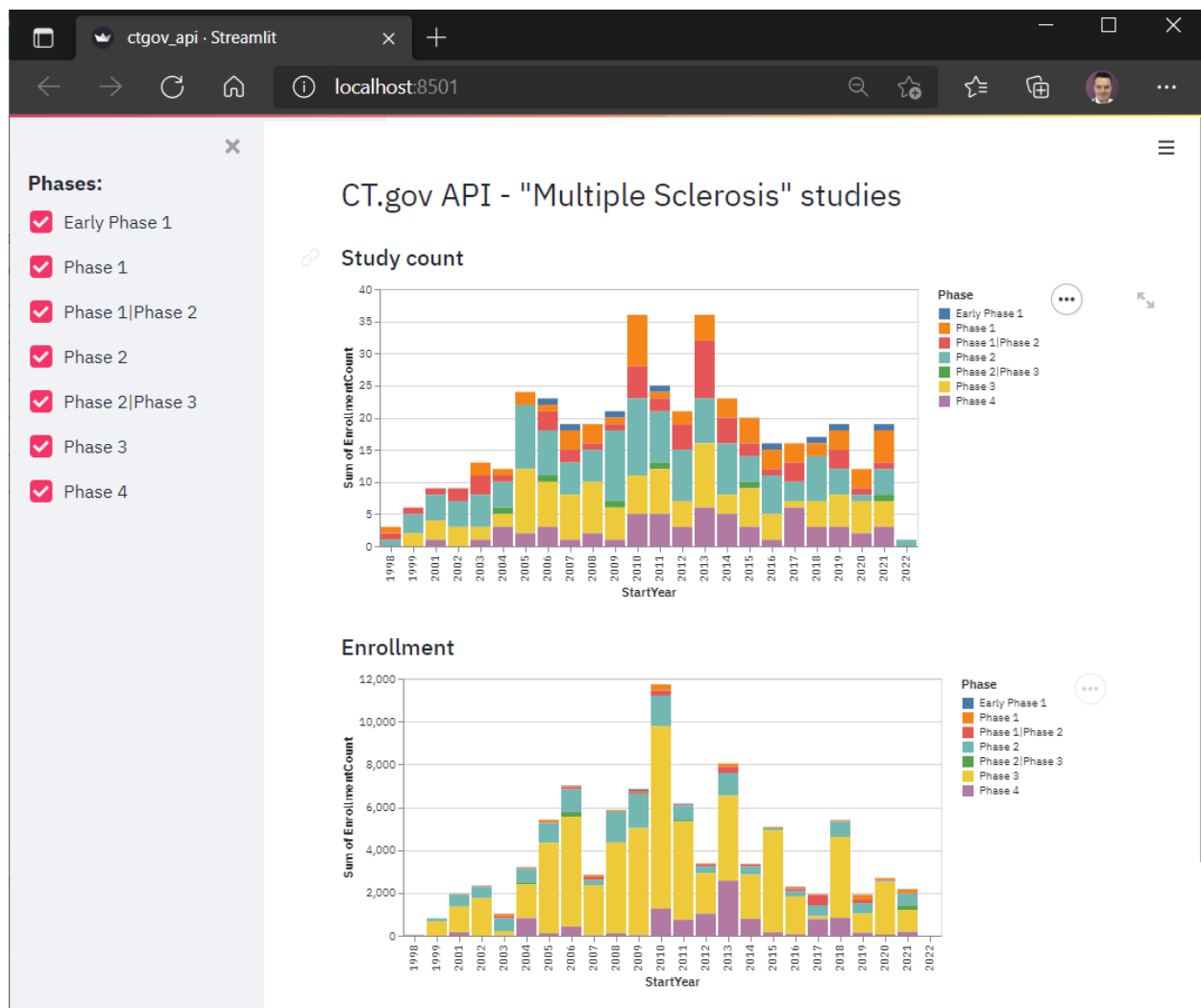
The portion of code below only shows the typical way of querying the API in Python and reading in the result as Pandas dataframe. For the full code to generate the example report, see Appendix 1.

CTgov_API.py

```
import requests, io
import pandas as pd

# Send and retrieve HTTP (REST) request
url = 'https://clinicaltrials.gov/api/query/study_fields?expr=multiple sclerosis&fields=NCTId%2CCondition%2Cphase%2CStartDate%2CEnrollmentCount&min_rnk=1&max_rnk=1000&fmt=csv'
res = requests.get(url).content

# Extract contents of request, skip CSV header (first 10 lines), to Pandas dataframe
data = pd.read_csv(io.StringIO(res.decode("utf-8")), skiprows=10).fillna(0)
```



...AND BEYOND

These are only three, relatively simple examples, however, they are hopefully a good illustration of what is possible when the power of the ClinicalTrials database accessed through its API and the analytical tools are combined. It is down to the programmer to decide if it is better to limit the data returned by specifying very clear and strict search expression, or if it may be of benefit to cast the API request net wider and focus on processing the results in the programming environment.

The three programs were written in R, SAS®, and Python to show that the API is language and system agnostic. Additional flexibility is provided by offering multiple formats of the data returned – XML, JSON, CSV, and in some scenarios TREE. Each will have advantages and disadvantages depending on the project and the language used. R and Python are well suited for dealing with all those formats. SAS, although technically capable of processing JSON and XML files, is however not easy to work with non-tabular data, which is why CSV may be the format of choice. Once the data is imported into the programming environment, there are numerous additional tools which can help process, visualize and beautify the results. There are no reasons why other languages couldn't be used. RShiny or Streamlit with Python can help create web interfaces and reports. Desktop applications can be build using Go or Java. NodeJS can take advantage of JSON to build an interface in the form of an Alexa skill, to tap into the registry using our voice. The options are in fact limitless.

OTHER RESOURCES AND TOOLS

It is worth mentioning that ClinicalTrials.gov API is not the only such tool available. Many services, websites, and databases have dedicated web interfaces. Some may require setting up an account and authenticating the requests, while others are open.

CDISC Library is one of the notable APIs available. The wealth of CDISC format information, which previously required perusing hundreds of pages of PDF documents, is now accessible for developers through a REST API. Of note, EMA does not offer any APIs to extract data from their EudraCT and EU CTR registries. They do, however, allow to download the contents in a non-automated way, which can then be processed by analytical tools.

CONCLUSIONS

Being able to take advantage of language and system agnostic REST APIs by querying them directly from the program, which subsequently processes and reports the results can be very helpful in automating processes, connecting in-house resources to public registries. Open access to public registries can bring value and new insights. ClinicalTrials.gov API offers great flexibility for both how we search the data, but also how it is returned. It is a free, resource with virtually no access limits or restrictions. The fact that the vast majority of clinical trials are reported in this registry and are being updated on an ongoing basis makes it an invaluable source of information for sponsors and researchers across the industry.

APPENDIX 1 (PYTHON, STREAMLIT EXAMPLE – FULL PROGRAM)

CTgov_API.py

```

import streamlit as st
import requests, io
import pandas as pd
import altair as alt

# Send and retrieve HTTP (REST) request
url = 'https://clinicaltrials.gov/api/query/study_fields?expr=multiple sclerosis&fields=NCTId%2CCCondition%2Cphase%2CStartDate%2CEnrollmentCount&min_rnk=1&max_rnk=1000&fmt=csv'
res = requests.get(url).content

# Extract contents of request, skip CSV header (first 10 lines), to Pandas dataframe
data = pd.read_csv(io.StringIO(res.decode("utf-8")), skiprows=10).fillna(0)
data = data[data['StartDate'] != 0]

# Get year from StartDate
data["StartYear"] = data["StartDate"].map(lambda dt: str(dt).split()[-1])

# Find all Phase values in dataframe
studyPhases = data["Phase"].unique().tolist()
studyPhases = [str(x) for x in studyPhases if x not in [0, "Not Applicable"]]
studyPhases.sort()

# List all Phase values as checkboxes in sidebar
st.sidebar.subheader("Phases:")
phases = {}
for phase in studyPhases:
    phases[phase] = st.sidebar.checkbox(phase, True)

# Subset dataframe to only phases selected in sidebar
subdata = data[data["Phase"].isin([k for k,v in phases.items() if v])]

# Sum up enrollment count by year and phase
studyCount = subdata.groupby(['StartYear', 'Phase'])['EnrollmentCount'].count().reset_index()
enrolmentCount = subdata.groupby(['StartYear', 'Phase'])['EnrollmentCount'].sum().reset_index()

# Altair study count chart
sc = alt.Chart(studyCount).mark_bar().encode(x='StartYear:O', y='sum(EnrollmentCount):Q', color='Phase:N')

# Altair enrollment chart
ec = alt.Chart(enrolmentCount).mark_bar().encode(x='StartYear:O', y='sum(EnrollmentCount):Q', color='Phase:N')

st.header('CT.gov API - "Multiple Sclerosis" studies')
st.subheader('Study count')
st.altair_chart(sc)
st.subheader('Enrollment')
st.altair_chart(ec)

```