phuse·share
connect·share·advance

APAC 2026

Implementing the CDISC Library RESTful API in R: Automated Access to Metadata Repositories and Controlled Terminologies

Jagadish Katam,
Princeps Technologies Inc.,

Disclaimer

- The views expressed in this presentation are solely those of the author and do not represent their employer or affiliated organizations. This content is for informational and educational purposes only and should not be considered legal, professional, or technical advice. While accuracy is strived for, technology and standards evolve. Users are encouraged to verify information and consult official CDISC documentation for current guidelines regarding the CDISC Library RESTful API and R implementation.

Motivation

Our goal is to demonstrate how to leverage the CDISC Library RESTful API to dynamically fetch and display metadata, creating an interactive and user-friendly interface for clinical data standards.

Agenda

01

CDISC Library Overview

Understanding the authoritative source for standards metadata

02

API Fundamentals

Core concepts of application programming interfaces

03

RESTful Architecture

How REST enables data exchange

04

HTTP Methods & Codes

Request types and response handling

05

Endpoints & Parameters

Navigating API structure

06

R Implementation

Practical coding examples

07

JSON Data Extraction

Processing API responses

08

Conclusions & Applications

Real-world benefits

What is the CDISC Library?

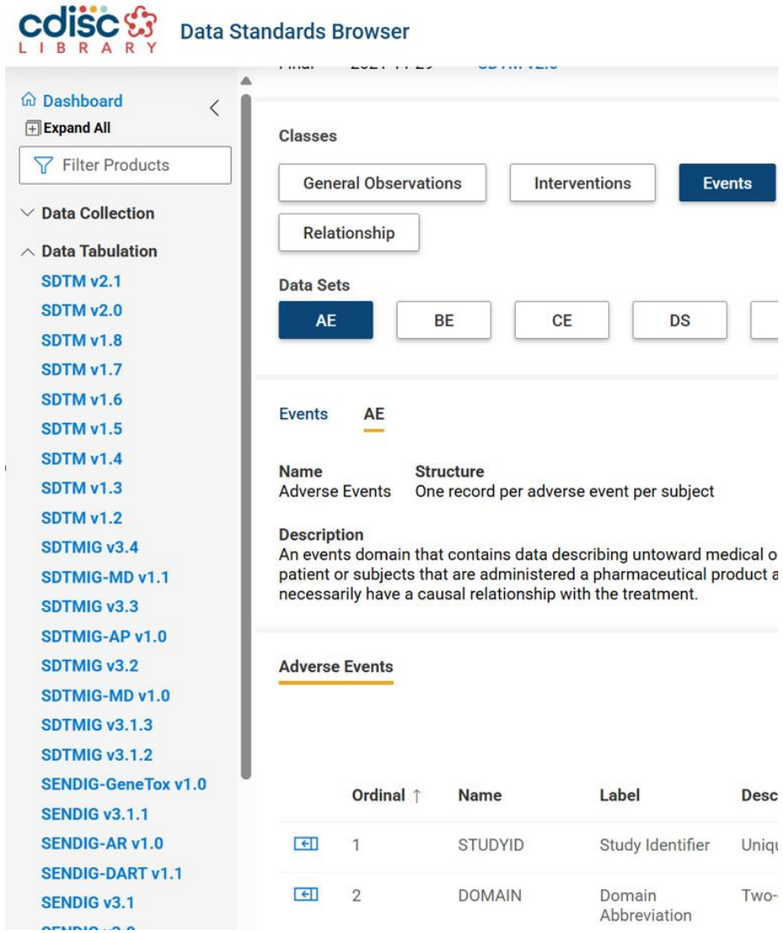
It provides comprehensive access to CDISC Data Standards metadata and Controlled Terminology in machine-readable formats.

Built on the **Microsoft Azure cloud platform**, this **metadata repository (MDR)** eliminates the need to parse static PDF documents, offering instead structured, programmatic access to critical standards information.

Core Components

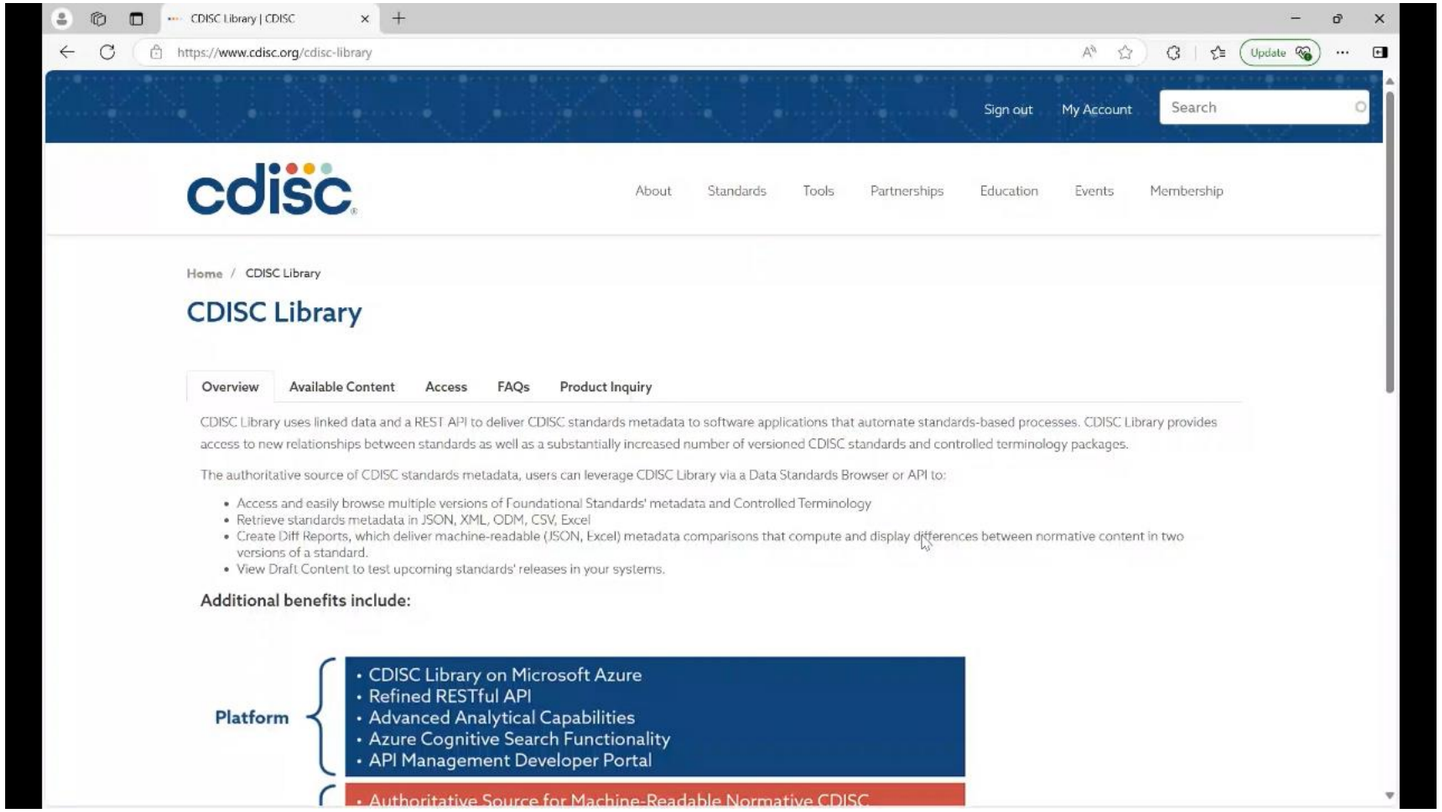
- **Data Standards Browser (DSB):** Interactive web interface for exploring standards
- **RESTful API:** Programmatic access for automated workflows

Available Standards: CDASH, SDTM, ADaM, QRS, Define-XML, Controlled Terminology, and more



Data Standards Browser (DSB)

Video demonstration: Navigating the Data Standards Browser



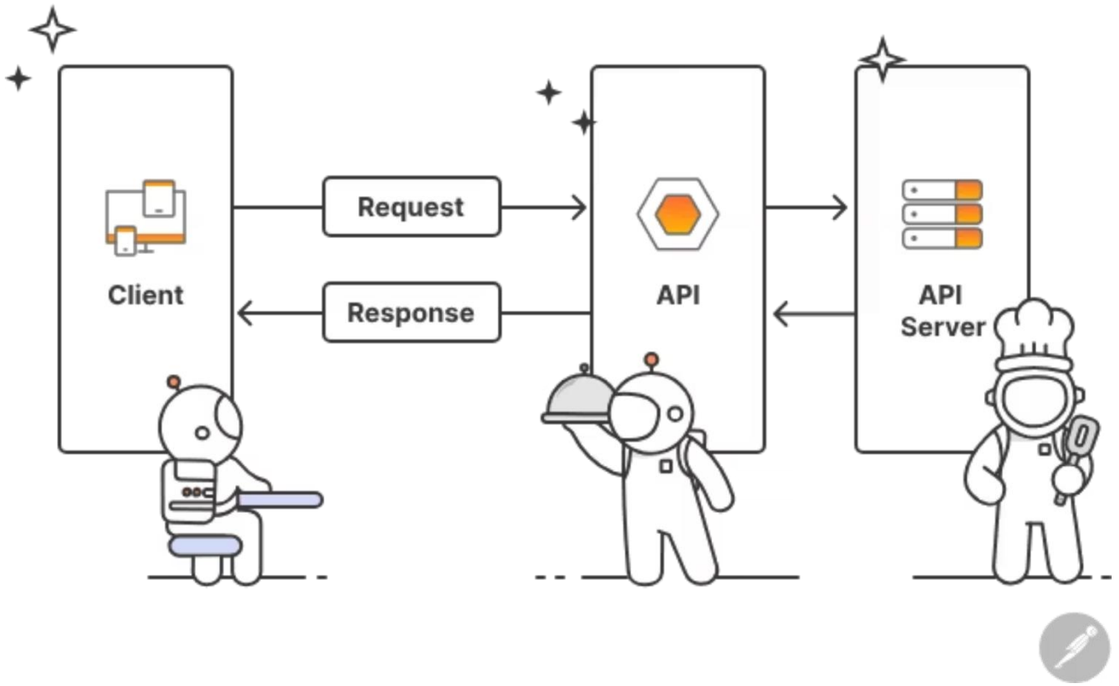
Understanding APIs

What is an API?

An **Application Programming Interface (API)** serves as a contract between client and server, enabling two applications to communicate seamlessly. When a client sends a request through the API, the server processes the action and returns a structured response.

Key API Categories

- **Private API:** Internal use within organizations
- **Public API:** Openly available to external developers
- **Partner API:** Shared between business partners



Source: https://www.postman.com/what-is-an-api/

What is REST?

- **REST (REpresentational State Transfer)** is an architectural style that defines a set of constraints for creating web services.
- It is a set of functions to which the developers perform requests and receive responses.
- In REST API, interaction is made via the HTTP protocol.
- REST also allows computers to talk to each other over a network.
- REST APIs return data in structured formats, primarily **JSON (JavaScript Object Notation)** or **XML**, making responses easy to parse and integrate into applications.

Understanding HTTP Status Codes

200 - OK
Request succeeded and server returned the requested data

201 - Created
New resource successfully created (used with POST requests)

400 - Bad Request
Invalid syntax or missing required parameters in the request

401 - Unauthorized
Authentication credentials are missing or invalid

403 - Forbidden
Authenticated but lacking required permissions for the resource

404 - Not Found
Requested resource does not exist at the specified endpoint

Retrieving Data with GET

The **GET** method is the most common HTTP verb, primarily used to request data from a specified resource. It is a fundamental operation for reading information from an API and should be *idempotent* (meaning multiple identical requests have the same effect as a single one).

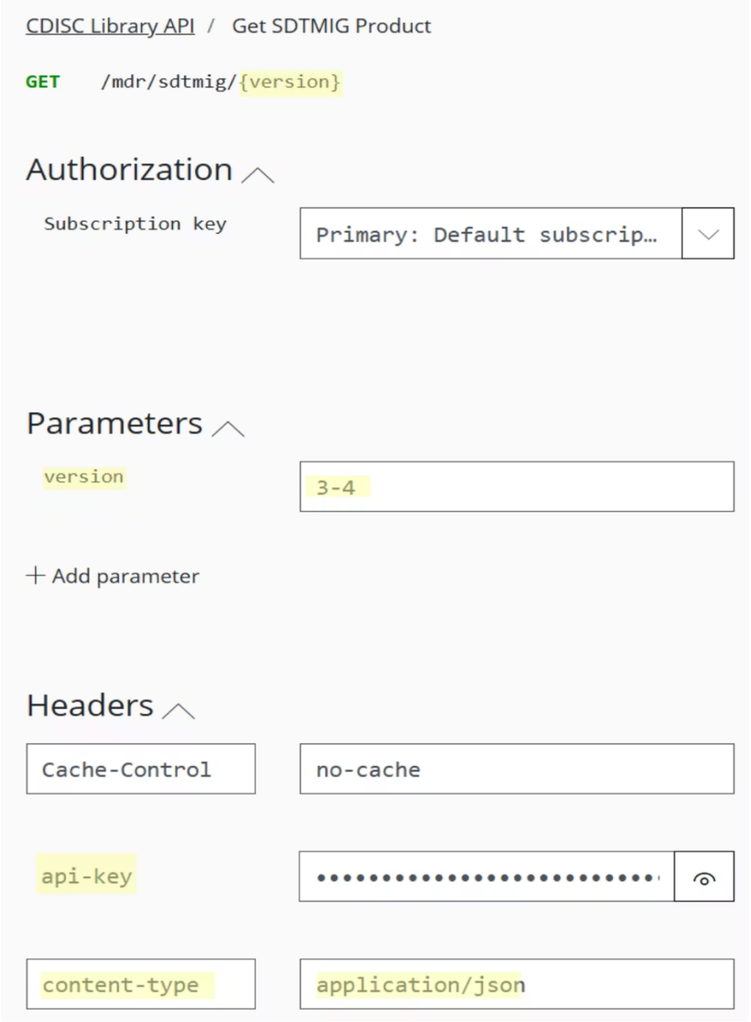
Anatomy of a GET Request URL

Example CDISC Library GET Request:

HTTP Methods: The GET Request

- These screenshots demonstrate a complete **GET request workflow** using the CDISC Library API.

- It illustrates the request structure, highlighting the base URL **(https://api.library.cdisc.org),** the endpoint path **(/api/mdr/sdtmig/3-4),** and essential headers including the API key and content-type.



HTTP Methods: Receive Response

- It displays the **JSON response** returned by the API, showcasing the nested data structure containing classes, datasets, and variables that will be parsed and transformed in R.

Endpoints and Parameters

An **endpoint** is a specific URL where an API can be accessed.

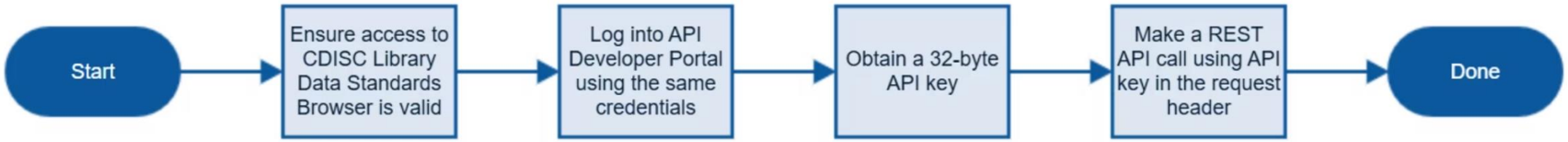
Parameters allow you to refine your requests and retrieve precisely the data you need.

Detailed Endpoint Example: SDTMig 3.4

- The `mdr` indicates that you are accessing the Metadata Repository.
- `sdtmig` specifies the Study Data Tabulation Model Implementation Guide (SDTMig) product.
- `3-4` indicates version 3.4 of the guide.
- `https://api.library.cdisc.org/api/mdr/sdtmig/3-4`

	PRODUCT	ENDPOINT	TITLE	TYPE
	All	All	All	All
11	ADaM	/mdr/adam/adamig-1-2	Analysis Data Model Implementation Guide Version 1.2	Implementation Guide
12	ADaM	/mdr/adam/adamig-1-3	Analysis Data Model Implementation Guide Version 1.3	Implementation Guide
13	SDTM	/mdr/sdtmig/3-1-2	Study Data Tabulation Model Implementation Guide: Human Clinical Trials Version 3.1.2 (Final)	Implementation Guide
14	SDTM	/mdr/sdtmig/3-1-3	Study Data Tabulation Model Implementation Guide: Human Clinical Trials Version 3.1.3 (Final)	Implementation Guide
15	SDTM	/mdr/sdtmig/3-2	Study Data Tabulation Model Implementation Guide: Human Clinical Trials Version 3.2 (Final)	Implementation Guide
16	SDTM	/mdr/sdtmig/3-3	Study Data Tabulation Model Implementation Guide: Human Clinical Trials Version 3.3 (Final)	Implementation Guide
17	SDTM	/mdr/sdtmig/3-4	Study Data Tabulation Model Implementation Guide: Human Clinical Trials	Implementation Guide
18	SDTM	/mdr/sdtmig/ap-1-0	Study Data Tabulation Model Implementation Guide: Associated Persons Version 1.0 (Final)	Implementation Guide
19	SDTM	/mdr/sdtmig/md-1-0	SDTM Implementation Guide for Medical Devices SDTMIG-MD 1.0 (Provisional)	Implementation Guide
20	SDTM	/mdr/sdtmig/md-1-1	Study Data Tabulation Model Implementation Guide for Medical Devices (SDTMIG-MD) Version 1.1	Implementation Guide

Accessing the CDISC Library API




```
# Load required library
library(httr2)

# API URL for SDTM IG version 3.4
url <- "https://api.library.cdisc.org/api/mdr/sdtmig/3-4"

# Construct the request with headers
req <- request(url) %>%
  req_headers(
    'Cache-Control' = 'no-cache',
    'api-key' = Sys.getenv("CDISC_LIBRARY_API_KEY"),
    'content-type' = 'application/json'
  )

# Send the request and fetch response
resp <- req %>% req_perform()

# Parse JSON response into R list
json_list <- resp %>% resp_body_json()
```

Understanding JSON Response Structure

When the API returns data, it arrives as a **JSON (JavaScript Object Notation)** file, a hierarchical structure of key-value pairs.

In R, this JSON is automatically parsed into a **nested list** structure.

JSON Components

- **Objects:** Named collections of key-value pairs
- **Arrays:** Ordered lists of values
- **Values:** Strings, numbers, booleans, or nested structures

Understanding this structure is crucial for extracting the specific data elements you need from the CDISC Library response.

```
> str(json_list)
List of 1
 $ _links:List of 7
  ..$ data-analysis  :List of 1
  .. ..$ _links:List of 2
  .. .. ..$ adam:List of 12
  .. .. .. ..$ :List of 3
  .. .. .. .. ..$ href : chr "/mdr/adam/adam-2-1"
  .. .. .. .. ..$ title: chr "Analysis Data Model Version 2.1"
  .. .. .. .. ..$ type : chr "Foundational Model"
  .. .. .. ..$ :List of 3
  .. .. .. .. ..$ href : chr "/mdr/adam/adam-adae-1-0"
  .. .. .. .. ..$ title: chr "Analysis Data Model Data Structure for Adverse Event Analysis Version 1.0"
  .. .. .. .. ..$ type : chr "Implementation Guide"
  .. .. .. ..$ :List of 3
  .. .. .. .. ..$ href : chr "/mdr/adam/adam-md-1-0"
  .. .. .. .. ..$ title: chr "ADaM Implementation Guide for Medical Devices v1.0 (ADaMIG-MD)"
  .. .. .. .. ..$ type : chr "Implementation Guide"
  .. .. .. ..$ :List of 3
  .. .. .. .. ..$ href : chr "/mdr/adam/adam-nca-1-0"
  .. .. .. .. ..$ title: chr "Analysis Data Model Implementation Guide for Non-compartmental Analysis"
  .. .. .. .. ..$ type : chr "Implementation Guide"
  .. .. .. ..$ :List of 3
  .. .. .. .. ..$ href : chr "/mdr/adam/adam-occds-1-0"
  .. .. .. .. ..$ title: chr "ADaM Structure for Occurrence Data (OCCDS) Version 1.0"
  .. .. .. .. ..$ type : chr "Implementation Guide"
  .. .. .. ..$ :List of 3
  .. .. .. .. ..$ href : chr "/mdr/adam/adam-occds-1-1"
  .. .. .. .. ..$ title: chr "ADaM Structure for Occurrence Data (OCCDS) Version 1.1"
  .. .. .. .. ..$ type : chr "Implementation Guide"
  .. .. .. ..$ :List of 3
  .. .. .. .. ..$ href : chr "/mdr/adam/adam-poppk-1-0"
  .. .. .. .. ..$ title: chr "Basic Data Structure for Population Pharmacokinetic (popPK) Analysis"
  .. .. .. .. ..$ type : chr "Implementation Guide"
  .. .. .. ..$ :List of 3
  .. .. .. .. ..$ href : chr "/mdr/adam/adam-tte-1-0"
  .. .. .. .. ..$ title: chr "ADaM Basic Data Structure for Time-to-Event Analyses Version 1.0"
  .. .. .. .. ..$ type : chr "Implementation Guide"
  .. .. .. ..$ :List of 3
  .. .. .. .. ..$ href : chr "/mdr/adam/adamig-1-0"
  .. .. .. .. ..$ title: chr "Analysis Data Model Implementation Guide Version 1.0"
  .. .. .. .. ..$ type : chr "Implementation Guide"
```

Use `str(json_list)` to explore the structure of your returned data

Extracting SDTM Classes and Datasets

- Once you have the **JSON** response loaded as a list in R, you can extract and transform it into structured data frames.

- The following code demonstrates how to iterate through the **nested structure** to create a tabular representation of SDTM classes and their associated datasets.

- The **%||%** operator provides default values when elements are NULL, ensuring robust data extraction even when optional fields are missing.

```r
# Load required libraries
library(purrr)

# Convert nested list into a data frame
df <- map_dfr(1:length(json_list$classes), function(i) {
  class_item <- json_list$classes[[i]]  # Extract class

  map_dfr(1:length(class_item$datasets), function(j) {
    dataset_item <- class_item$datasets[[j]]  # Extract
dataset

    # Create data frame with key fields
    data.frame(
      class = class_item$label,
      datastructure = dataset_item$datasetStructure %||% NA,
      description = dataset_item$description %||% NA,
      label = dataset_item$label %||% NA,
      name = dataset_item$name %||% NA,
      ordinal = dataset_item$ordinal %||% NA,
      stringsAsFactors = FALSE
    )
  })
})
```

SDTM Implementation Guide Structure

- The extracted data reveals the complete organizational structure of **SDTM Implementation Guide version 3.4**.
- It including all domain classes such as **Special Purpose, Interventions, Events, Findings, Trial Design,** and **Relationship datasets**.
- Each dataset includes its name, label, description, and structural information.

Show 10 entries Search:

	CLASS	DATASTRUCTURE	DESCRIPTION	LABEL	NAME	ORDINAL
	All	All	All	All	All	All
1	General Observations					
2	General Observations					
3	Interventions	One record per recorded intervention occurrence per subject	An interventions domain that contains the agents administered to the subject as part of a procedure or assessment, as opposed to drugs, medications and therapies administered with therapeutic intent.	Procedure Agents	AG	2
4	Interventions	One record per recorded intervention occurrence or constant-dosing interval per subject	An interventions domain that contains concomitant and prior medications used by the subject, such as those given on an as needed basis or condition-appropriate medications.	Concomitant/Prior Medications	CM	6
5	Interventions	One record per protocol-specified study treatment, collected-dosing interval, per subject, per mood	An interventions domain that contains information about protocol-specified study treatment administrations, as collected.	Exposure as Collected	EC	15

Showing 1 to 5 of 65 entries Previous 1 2 3 4 5 … 13 Next

Extracting Dataset Variables

```
# Convert list of lists into a data frame
dataset_df <- map_dfr(1:length(json_list$classes), function(i) {
  class_data <- json_list$classes[[i]]

  map_dfr(1:length(class_data$datasets), function(j) {
    dataset <- class_data$datasets[[j]]

    if (is.null(dataset$datasetVariables)) {
      return(NULL)  # Skip datasets with no variables
    }

    # Extract dataset variables
    variable_df <- map_dfr(1:length(dataset$datasetVariables), function(x) {
      var <- dataset$datasetVariables[[x]]

    # Extract codelist href if available, otherwise NA
      href_value <- if (!is.null(var$`_links`$codelist) && length(var$`_links`$codelist) > 0) {
        var$`_links`$codelist[[1]]$href %||% NA
      } else {
        NA
      }

      data.frame(
        dataset = dataset$name,
        Ordinal = as.numeric(var$ordinal) %||% NA,
        Name = var$name %||% NA,
        Label = var$label %||% NA,
        Description = var$description %||% NA,
        Datatype = var$simpleDatatype %||% NA,
        Role = var$role %||% NA,
        core = var$core %||% NA,
        Codelist = stringr::str_extract(href_value,'C\\d+$'),
        stringsAsFactors = FALSE
      )
    })

  })
}) |> arrange(dataset,Ordinal)
```

Input Structure

The program expects an input object:

- **json_list**
  - **classes (list)**
    - **datasets (list)**
      - **datasetVariables (list)**

Each datasetVariable contains metadata such as name, label, datatype, role, ordinal position, and optional codelist links.

Extracted Dataset Structure

- This screenshot displays the resulting data frame, **`dataset_df`,** which contains structured SDTM variable metadata extracted from the API response using the R code shown previously.

Show 3 entries Search:

	DATASET	ORDINAL	NAME	LABEL	DESCRIPTION	DATATYPE	ROLE	CORE	CODELIST_HREF
	All	All	All	All	All	All	All	All	All
1	AE	1	STUDYID	Study Identifier	Unique identifier for a study.	Char	Identifier	Req	
2	AE	2	DOMAIN	Domain Abbreviation	Two-character abbreviation for the domain.	Char	Identifier	Req	
3	AE	3	USUBJID	Unique Subject Identifier	Identifier used to uniquely identify a subject across all studies for all applications or submissions involving the product.	Char	Identifier	Req	

Showing 1 to 3 of 1,917 entries Previous 1 2 3 4 5 ... 639 Next

Accessing Controlled Terminology

Controlled Terminology (CT) ensures consistent coding of clinical data across studies.

The CDISC Library API provides programmatic access to the latest CT packages, including all codelists and term definitions.

```
# API URL for SDTM CT package
url <- "https://api.library.cdisc.org/api/mdr/ct/packages/sdtmct-2024-09-27"

# Construct the request
req <- request(url) %>%
 req_headers(
 'Cache-Control' = 'no-cache',
 'api-key' = Sys.getenv("CDISC_LIBRARY_API_KEY"),
 'content-type' = 'application/json'
 )

# Send request and parse response
resp <- req %>% req_perform()
ct_list <- resp %>% resp_body_json()
```

Processing Controlled Terminology Data

```
# Get length of codelists
codelist_count <- length(ct_list$codelists)

# Convert all nested lists into a data frame
ct_codelist_df <- map_dfr(1:codelist_count, function(i) {

  # Extract current codelist
  ctcodelist <- ct_list$codelists[[i]]

  # Extract codelist-level details
  ct_codelist_info <- data.frame(
    codelist = ctcodelist$conceptId %||% NA,
    definition = ctcodelist$definition %||% NA,
    extensible = ctcodelist$extensible %||% NA,
    name = ctcodelist$name %||% NA,
    nci_preferred_Term = ctcodelist$preferredTerm %||% NA,
    submission_Value = ctcodelist$submissionValue %||% NA,
    stringsAsFactors = FALSE
  )

  # Get length of terms
  terms_count <- length(ctcodelist$terms)

  # Extract term-level details (if available)
  terms_df <- map_dfr(1:terms_count, function(j) {
    term <- ctcodelist$terms[[j]]
    data.frame(
      term = term$conceptId %||% NA,
      term_definition = term$definition %||% NA,
      term_nci_preferred_Term = term$preferredTerm %||% NA,
      term_submission_Value = term$submissionValue %||% NA,
      stringsAsFactors = FALSE
    )
  })

  # Merge codelist details with terms (repeat codelist info for each term)
  bind_cols(ct_codelist_info, terms_df)
})
```

- After retrieving the **Controlled Terminology (CT)** package from the API, this code transforms its nested **JSON** structure into a flat, tabular data frame.
- It iterates through each **codelist** and its associated terms, extracting both **codelist-level metadata** (concept ID, definition, and extensibility) and term-level details (submission values, preferred terms).
-
- The result is a single comprehensive data frame, ideal for downstream analysis and reference.

Controlled Terminology Data

- This screenshot displays the resulting data frame after processing the **Controlled Terminology (CT)** package, showcasing the flattened CT structure.

Show 10 entries Search:

	CODELIST	DEFINITION
	All	All
1	C141657	10-Meter Walk/Run test code.
2	C141657	10-Meter Walk/Run test code.
3	C141657	10-Meter Walk/Run test code.
4	C141657	10-Meter Walk/Run test code.
5	C141656	10-Meter Walk/Run test name.
6	C141656	10-Meter Walk/Run test name.
7	C141656	10-Meter Walk/Run test name.
8	C141656	10-Meter Walk/Run test name.
9	C141663	4-Stair Ascend test code.
10	C141663	4-Stair Ascend test code.

Showing 1 to 10 of 42,831 entries Previous 1 2 3 4 5 … 4,284 Next

Key Benefits and Conclusions

Direct Standards Access

Retrieve the latest official CDISC standards (SDTM, ADaM, Define-XML) programmatically without manual downloads

Enforced Standardization

Ensure consistent implementation of data structures across clinical trials with automated validation

Error Reduction

Minimize manual transcription errors by programmatically importing standards into your workflows

Structured Responses

Work with JSON instead of static PDFs, enabling seamless integration with R and other tools

Programmatic Integration

Build automated pipelines that stay synchronized with updated CDISC standards releases

Comprehensive Metadata

Access dataset structures, variable definitions, and controlled terminology in a single unified interface

References

- Aerts, J. (2019). *Implementing the CDISC Library API in software applications: First experiences.* Presented at **PHUSE EU Connect 2019**, Tarrenz, Austria.

- Hinson, J. (2021). *How the R programming language handles the CDISC Library API.* Presented at **PHUSE US Connect 2021**.

- Radelicki, R. (2020). *CDISC Library try-out: From implementation to evaluation of the API.* Presented at **PHUSE US Connect 2020**, Mechelen, Belgium.

- Jansen, L. (2021). *Extracting data standards metadata and controlled terminology from the CDISC Library using SAS© with PROC LUA* (Paper AD-168). Presented at **PharmaSUG 2021.**

Thank You!

Connect with me on LinkedIn

Explore the CDISC Web Portal