Paper AD16

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

Jagadish Katam, Princeps Technologies Inc., Hyderabad, India

ABSTRACT

The CDISC Library API offers programmatic access to clinical data standards, such as SDTM, ADaM domains, variables, and controlled terminologies enabling automation of compliance workflows in clinical research. This presentation explores an R-based implementation of the API to retrieve and structure CDISC metadata and terminologies, eliminating manual extraction from static resources like PDFs or Excel files. Using the httr2, jsonlite, and tidyverse packages, we demonstrate a reproducible pipeline that queries the /mdr/ct/packages endpoint, parses nested JSON responses into structured data frames, and merges codelists with their terms for downstream analysis and review. To make the extracted information accessible, we further integrate this pipeline into a Shiny web application. The Shiny app allows users to explore CDISC standards dynamically through searchable and filterable tables, download options, and domain-specific metadata views. It supports real-time querying of the API and presents users with an interface to browse codelists without requiring manual API interaction.

INTRODUCTION

CDISC standards are a foundational element of regulatory-grade clinical data processing. SDTM supports standardized tabulation datasets, ADaM supports analysis-ready datasets, and CDISC Controlled Terminology provides submission-consistent, semantically harmonized values. Regulatory authorities increasingly expect sponsors to demonstrate not only conformance of delivered datasets but also traceability to standards versions and governance decisions.

In many organizations, CDISC implementation remains document-driven. Programmers and standards leads frequently depend on:

- PDF Implementation Guides (IGs) for SDTM/ADaM rules and variable expectations
- Excel-based internal standards libraries and mapping specifications
- Manually maintained controlled terminology extracts

This approach introduces recurring risks:

- **Interpretation risk:** narrative text is interpreted differently across teams and vendors
- **Version drift:** the “current” standard version may change between studies, milestones, or vendors
- **Manual effort:** repeated extraction of dataset/variable lists and codelists is time-consuming
- **Limited automation:** compliance checks may rely on ad hoc checks rather than governed logic
- **Traceability gaps:** difficult to prove which standard metadata was used at a given time

The CDISC Library addresses these challenges by providing a cloud-based metadata repository where standards content is also exposed as structured, machine-readable JSON through a RESTful API. This enables R (and other languages) to retrieve metadata directly from the authoritative source and incorporate it into reproducible programming workflows.

This paper focuses on a practical implementation in R, aligned to the provided presentation agenda: CDISC Library overview, REST fundamentals, HTTP methods, endpoints/parameters, R-based API requests, JSON parsing, metadata extraction, and controlled terminology retrieval.

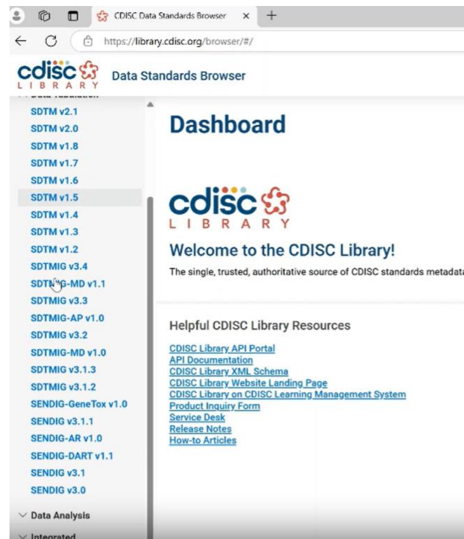
OVERVIEW OF THE CDISC LIBRARY API

PURPOSE AND CAPABILITIES

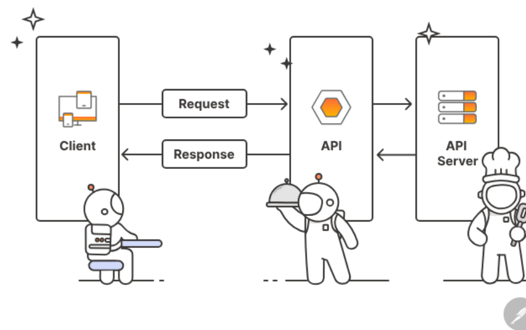
The CDISC Library is a single, trusted, authoritative source of CDISC Data Standards metadata and Controlled Terminology. It is implemented as a cloud-based metadata repository (MDR) on the Microsoft Azure platform.

The CDISC Library is composed of:

- Data Standards Browser (DSB)



- API



The DSB supports interactive browsing, while the API supports programmatic access for automation, integration, and governance. Users can browse and retrieve metadata such as CDASH, SDTM, ADaM, QRS, and Controlled Terminology. CDISC implementation guides remain available in PDF format, and the same contents are also available in machine-readable formats.

Through the API, users can query detailed metadata for domains, variables, value-level metadata, controlled terminology, and relationships without relying on static documents.

SUPPORTED STANDARDS AND VERSIONS

The API supports multiple CDISC standards, including but not limited to:

- SDTM (Study Data Tabulation Model)
- ADaM (Analysis Data Model)
- CDASH (Clinical Data Acquisition Standards Harmonization)
- SEND (Standard for Exchange of Nonclinical Data)
- Controlled Terminology (CT)

Each standard is available across multiple published versions, allowing users to explicitly request metadata aligned with a specific regulatory submission baseline.

Architecture and API Design

REST PRINCIPLES

The CDISC Library API follows standard RESTful design principles:

- Resource-oriented URLs
- Stateless requests
- Use of standard HTTP methods (GET primarily)
- Clear separation of resources such as standards, domains, and terminology

This design makes the API accessible from a wide range of programming environments and tools.

AUTHENTICATION AND ENDPOINTS

Access to the API requires authentication using an API key issued by CDISC. Authentication is typically handled via HTTP headers, ensuring secure and auditable access.

Endpoints are structured hierarchically, reflecting the logical organization of CDISC standards. For example, users can navigate from a standard to a specific version, domain, and variable.

HTTP METHODS, CODES, AND OPERATIONAL EXPECTATIONS

The presentation highlights the HTTP GET method and common HTTP response codes. In the context of automation:

- A **200 OK** is required before attempting to parse JSON.
- A **401 Unauthorized** strongly indicates missing/invalid API key.
- A **400 Bad Request** indicates invalid/missing parameters (e.g., malformed version).
- A **404 Not Found** is typically a wrong endpoint or unavailable version/package.

In regulated environments, these codes should be logged with sufficient context (URL requested, timestamp, standard version), as they support reproducibility and issue triage.

DATA FORMATS AND RESPONSES

API responses are returned in JSON format, making them well-suited for downstream parsing and transformation. The structured nature of JSON responses allows metadata to be easily mapped into tabular structures for validation, reporting, or rule-based processing.

- CDISC API GET request workflow

CDISC Library API / Get SDTMIG Product

GET /mdr/sdtmig/{version}

Authorization

Subscription key: Primary: Default subscrip...

Parameters

version: 3-4

+ Add parameter

Headers

Cache-Control: no-cache

api-key: ••••••••••••••••••••••••••

content-type: application/json

HTTP request

```
GET https://api.library.cdisc.org/api/mdr/sdtmig/3-4 HTTP/1.1

Cache-Control: no-cache
api-key: ••••••••••••••••••••••••••••••
content-type: application/json
```

HTTP response

```
HTTP/1.1 200 OK

content-length: 2297156
content-type: application/json
date: Wed, 05 Feb 2025 10:26:31 GMT
strict-transport-security: max-age=31536000; includeSubDomains
vary: Origin
x-ms-middleware-request-id: 62eec67c-2616-4d8b-9149-e212903f7e2d
x-referenceid: 051ce3ce-d77c-484d-b6c2-30646942a11e
```


```
{
    "_links": {
        "model": {
            "href": "/mdr/sdtm/2-0",
            "title": "Study Data Tabulation Model Version 2.0",
            "type": "Foundational Model"
        },
```


KEY API ENDPOINTS AND USE CASES

METADATA RETRIEVAL

Core endpoints allow retrieval of:

- Domain-level metadata (e.g., SDTM DM, AE)
- Variable definitions, roles, and attributes
- Value-level metadata and codelists

This enables programmers to dynamically build expectations for dataset structures directly from the standard.

VERSION CONTROL

The API allows explicit specification of standard versions, which is critical for:

- Maintaining submission traceability
- Supporting legacy studies
- Comparing metadata across versions

Version-aware queries reduce ambiguity and prevent unintended drift from approved standards.

CONTROLLED TERMINOLOGY

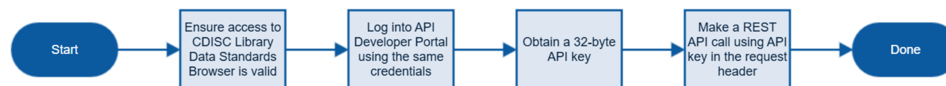
Controlled terminology endpoints provide access to:

- Codelists
- Permissible values
- Definitions and submission values
-

This supports automated terminology checks and alignment with published CDISC CT releases.

	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

Using the CDISC Library API with R



REQUIRED R PACKAGES

The presentation demonstrates how R can serve as an effective client for the CDISC Library API. Commonly used packages include:

- *httr2* for HTTP requests and authentication
- *jsonlite* for parsing JSON responses
- *dplyr* and *tidyr* for metadata transformation
- purrr for iterating through nested JSON objects
- stringr for extracting codelist identifiers from URLs

These packages are widely adopted in regulated analytics environments.

In practice, the following are typical imports:

```
r

library(httr2)
library(purrr)
library(dplyr)
library(stringr)
```

IMPLEMENTATION APPROACH: REQUEST, PERFORM, PARSE, TRANSFORM

A regulated-friendly API client flow should follow four phases:

1. **Request construction** (URL, headers, content type)
2. **Request execution** (GET via req_perform())
3. **Response qualification** (status code checks; error handling)
4. **JSON parsing and transformation** (list → flattened data frame)

EXAMPLE: RETRIEVING SDTM METADATA

This example implements the presentation endpoint for SDTMIG 3-4:

Step 1: Define endpoint and headers

```
r

library(httr2)

url <- "https://api.library.cdisc.org/api/mdr/sdtmig/3-4"

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

Explanation (why this matters for production use):

- *Sys.getenv("CDISC_LIBRARY_API_KEY") avoids hardcoding secrets into code repositories.*
- *content-type header indicates the expected payload format for the interaction.*

- *Cache-Control reflects the slide example and can be useful when fresh retrieval is required.*

Step 2: Perform request and parse JSON

```
r

resp <- req %>% req_perform()

# Optional safety check before parsing
status <- resp_status(resp)
if (status != 200) {
  stop("CDISC Library request failed with HTTP status: ", status)
}

json_list <- resp %>% resp_body_json()
```

Explanation (JSON parsing considerations):

- resp_body_json() returns a nested R list.
- Parsing should only occur when status indicates success (typically 200).
- In automated pipelines, error messages should include the URL and standard version to support traceability.

```
> 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"
```

Step 3: Understand the JSON hierarchy
The presentation describes that JSON is loaded into R as a list of nested objects. For SDTMIG metadata, an important structural expectation is:

- Top-level includes classes
- Each class contains datasets
- Each dataset may contain datasetVariables
- Variables may contain _links including codelist references

Understanding this hierarchy determines how to flatten it.

Step 4: Extract SDTMIG classes and datasets into a data frame
The presentation code uses nested map_dfr() to traverse classes and datasets.

```
r

library(purrr)
library(dplyr)
```

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

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

    # Create a data frame with required 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
    )
  })
})
```

Explanation (why map_dfr() is appropriate):

- map_dfr() iterates and row-binds results into a single data frame.
- Nested map_dfr() mirrors the JSON nesting: outer loop over classes, inner loop over datasets.
- %||% NA provides defensive handling for missing fields (common in variable metadata where not all datasets share all attributes).

	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

Resulting data utility:
The resulting df supports:

- Standard-driven dataset inventory creation
- Programmatic documentation (e.g., producing tables of SDTMIG datasets by class)
- Standards governance comparisons (e.g., compare SDTMIG versions by dataset count)

Example: Extract SDTMIG dataset-variable metadata
The presentation includes extracting SDTMIG datasets and variables, including codelist identifiers embedded in _links. This section expands on how that extraction can support compliance checks.

Step 1: Extract dataset-variable metadata

```
r

# 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)
```

Explanation (key technical points):

- **Null handling:** SDTMIG metadata may include datasets that do not list variables at the location expected, so is.null() checks prevent failures.
- **Codelist extraction:** codelist identifiers are embedded in hyperlink references; extracting C#### enables controlled terminology joins later.
- **Ordering:** Sorting by dataset and ordinal mimics how specifications are typically presented, which supports reviewer-friendly outputs.

	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

Controlled Terminology API Request

Controlled terminology is exposed through a dedicated endpoint that references a specific terminology package and release date. In the example shown in the presentation, the SDTM controlled terminology package released on 27-Sep-2024 is retrieved.

```
r

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

# Parse JSON response
ct_list <- resp %>% resp_body_json()
```

Structure of the Controlled Terminology JSON Response

Once loaded into R, the controlled terminology response is represented as a nested list. The top-level object contains a collection of **codelists**, each of which includes:

- Codelist-level metadata (concept ID, name, definition, extensibility)
- A nested list of **terms** associated with the codelist
- Term-level metadata including submission values and preferred terms

Conceptually, the JSON hierarchy follows this structure:

- codelists
    - codelist metadata
    - terms
        - term metadata

Understanding this hierarchy is essential for correctly flattening the JSON into a tabular format suitable for validation and reporting.

Extracting Controlled Terminology from JSON

The presentation demonstrates flattening the nested JSON structure into a single data frame containing both codelist-level and term-level attributes.

```
r

library(purrr)
library(dplyr)

codelist_count <- length(ct_list$codelists)

ct_codelist_df <- map_dfr(seq_len(codelist_count), function(i) {

  ctcodelist <- ct_list$codelists[[i]]

  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
  )

```

```
terms_df <- map_dfr(seq_along(ctcodelist$terms), 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
    )
  })

  bind_cols(ct_codelist_info, terms_df)
})
```

Explanation of the Extraction Logic

This extraction process follows a consistent and auditable pattern:

1. **Iterate over codelists**
   Each codelist is processed independently, ensuring no assumptions are made about list length or ordering.
2. **Capture codelist-level metadata**
   Attributes such as concept ID, extensibility, and submission value are repeated for each associated term to preserve relational context.
3. **Iterate over terms within each codelist**
   Each term's concept ID, preferred term, definition, and submission value are extracted.
4. **Flatten hierarchical data**
   bind_cols() is used to combine codelist metadata with term metadata, producing a denormalized structure suitable for analysis.
5. **Defensive handling of missing values**
   The %||% NA pattern ensures robustness when optional attributes are absent.
6. The resulting data frame represents a complete, structured view of controlled terminology content for the specified release.

CDISC SDTM WEB PORTAL – SUMMARY

The CDISC SDTM Web Portal is an interactive web application designed to provide browser-based access to CDISC standards metadata and controlled terminology, particularly for SDTM (Study Data Tabulation Model) and ADaM (Analysis Data Model) standards. The application appears to be built using R Shiny and hosted on shinyapps.io, a platform for deploying interactive R web apps to the cloud.

- **Link to access the CDISC Library Web Portal**

CONCLUSION

The CDISC Library API represents a significant advancement in how clinical data standards are accessed and operationalized. By exposing authoritative CDISC metadata through a RESTful interface, the API enables automation, reproducibility, and stronger standards governance. When combined with R, the API supports metadata-driven programming and scalable compliance checks aligned with regulatory expectations. As clinical development continues to evolve toward more automated and data-centric workflows, programmatic standards access will become increasingly essential. The CDISC Library API provides a practical and extensible foundation for this future.

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.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Jagadish Katam
Company: Princeps Technologies Inc.,
Address: Hyderabad, India
Email: jagadish.katam@princepstech.com
Website: https://www.princepstech.com/

Brand and product names are trademarks of their respective companies.