

Extracting Data Standards Metadata and Controlled Terminology from the CDISC Library using SAS® with PROC LUA

Lex Jansen, SAS Institute Inc.

ABSTRACT

The CDISC Library is the single, trusted, authoritative source of CDISC Data Standards metadata and Controlled Terminology. It uses linked data and a REST API to deliver CDISC Data Standards metadata and Controlled Terminology in a machine-readable format to software applications that automate standards-based processes.

This paper shows how metadata can be extracted in SAS through the CDISC Library API using a REST API request with PROC HTTP. PROC LUA will be used in SAS to manage the PROC HTTP requests. PROC LUA will also be used to parse JavaScript Object Notation (JSON) response strings with a JSON library to extract data before storing them in SAS data sets.

The last part of the paper will focus on loading the extracted data sets into the Data Standards and Controlled Terminology areas of the Clinical Management module of SAS Life Science Analytics Framework.

WHAT IS THE CDISC LIBRARY

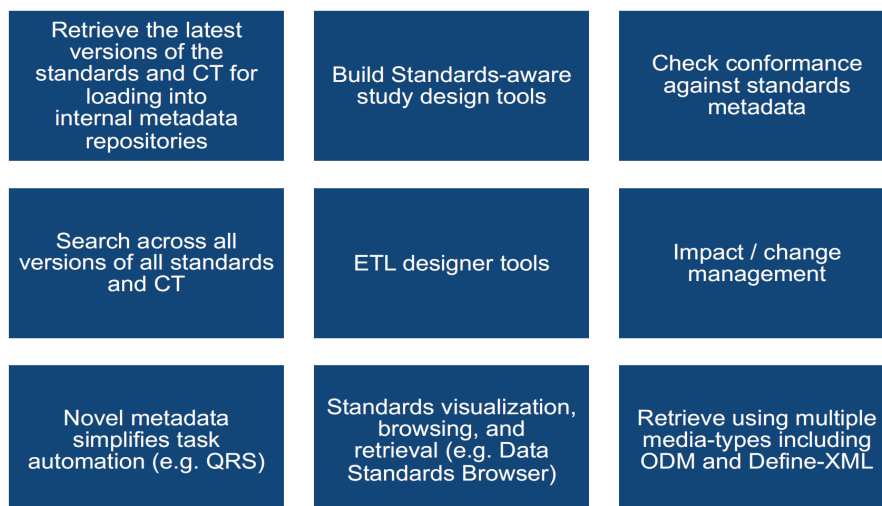
The CDISC Library is a single, trusted, authoritative source of CDISC Data Standards metadata and Controlled Terminology. It is a cloud-based metadata repository (MDR) on the Microsoft Azure platform. The CDISC Library uses linked data and a REST API to deliver CDISC Data Standards metadata and Controlled Terminology in a machine-readable format to software applications that automate standards-based processes. It also provides access to new relationships between standards

The CDISC Library is available to CDISC Members and Open Source Developers. An account can be requested at <https://www.cdisc.org/cdisc-library>

You will need a separate CDISC Library account, which is different from the CDISC account to access the Members Only Area on the CDISC website.

The CDISC Library has many uses cases. In a recent CDISC webinar Sam Hume and Anthony Chow mentioned the use cases as mentioned in Figure 1 [1].

Figure 1. Common use case for the CDISC Library



There are 3 different ways of accessing the CDISC Library:

- Manual downloads from the **CDISC Library Archives**
- **Data Standards Browser** – including downloads
- Programmatically through the **CDISC Library REST API**

All of these require a CDISC account which can be created at [cdisc.org](https://www.cdisc.org). For the Data Standards Browser and the CDISC Library REST API you will need a separate CDISC Library account.

CDISC Library Archives

Formerly known as CDISC SHARE Exports, the CDISC Library Archives allow you to download:

- Machine-readable metadata published from the CDISC Library. (Excel, CSV, ODM XML, Define-XML, RDF)
- PDF documents (a standard's specifications, implementation guides (IG), or user guides (UG))
- Diff (Difference) files that summarize changes between standards

The CDISC Library Archives can be access at <https://www.cdisc.org/members-only/cdisc-library-archives>

Figure 2 shows an example of files that can be downloaded.

Figure 2. Downloading Files from the CDISC Library Archives

SDTM				
Date Posted to Archive	Content	Version	Type	Download Files
SDTMIG 3.3				
2018-11-19	SDTM IG	3.3	Metadata	Excel
2018-11-19	SDTM Diff	3.3 - 3.2	Metadata	Excel
2020-06-26	SDTM IG	3.3	Document	PDF
SDTM 1.7				
2018-11-19	SDTM	1.7	Metadata	Excel
2018-11-19	SDTM Diff	1.7 - 1.6	Metadata	Excel
SDTM Terminology				
2021-01-21	SDTM Terminology	2020-12	Metadata	ODM v1.3.1 RDF Excel Diff
2020-11-25	SDTM Terminology	2020-11	Metadata	ODM v1.3.1 RDF Excel Diff

Data Standards Browser

Data Standards Browser: <https://library.cdisc.org/browser>. Here you can browse Data Standards and Controlled Terminology and see relations between them. The Data Standards Browser also allows you to manually download Data Standards and Controlled Terminology as Comma Separated Values (CSV) and Microsoft Excel (XLSX).

Recently a capability was added to download difference reports between versions of Data Standards or Controlled Terminology. These difference reports can be downloaded as Microsoft Excel files (XLSX).

Figure 3. Data Standards Browser - Dashboard

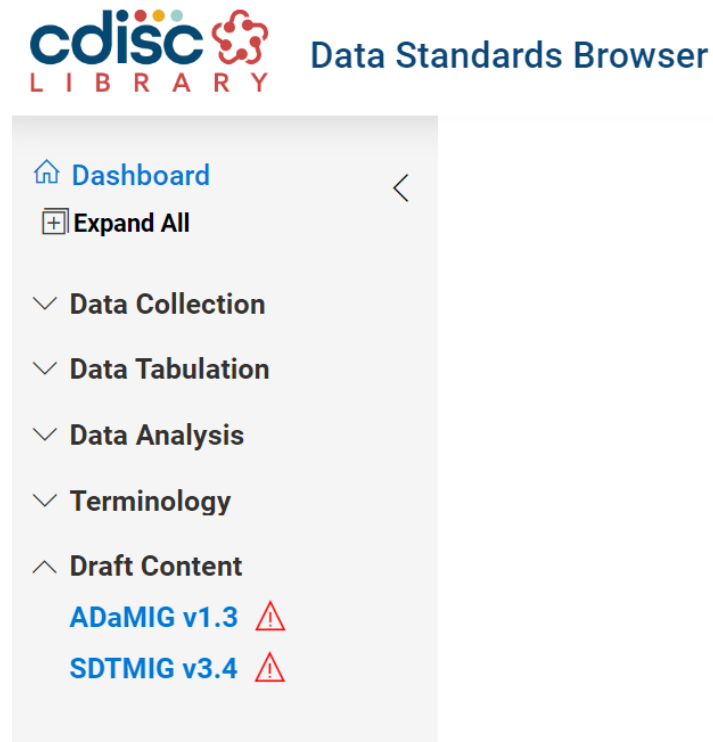


Figure 4 show an example of browsing the SDTM-IG version 3.3 and Figure 5 show the use of the search capability..

Figure 4. Data Standards Browser – Browsing SDTM-IG 3.3

The screenshot shows the CDISC Library Data Standards Browser interface. On the left is a sidebar with a navigation menu including 'Dashboard', 'Expand All', 'Data Collection', 'Data Tabulation', and 'Data Analysis'. The main content area is titled 'Classes' and features tabs for 'General Observations', 'Special-Purpose' (selected), 'Interventions', 'Events', 'Findings', 'Findings About', and 'Trial Data'. Below these are 'Data Sets' (CO, DM, SE, SM, SV) and a 'Special-Purpose' section for the 'DM' domain. The 'DM' section includes a 'Name' (Demographics), a 'Structure' (One record per subject), and a 'Description' (A special-purpose domain that includes a set of essential standard variables that describe each subject in a clinical study. It is the parent domain for all CDISC Controlled Terminology, DOMAIN, C49572, 2018-06-29). Below this is a 'Demographics' table with columns: Ordinal, Name, Label, Description, Data Type, Role, and Core. The table lists three items: STUDYID, DOMAIN, and USUBJID. On the right, a 'Details' panel shows information for the selected 'USUBJID' domain, including its Ordinal (3), Name (USUBJID), Label (Unique Subject Identifier), Description (Identifier used to uniquely identify a subject...), Data Type (Char), Role (Identifier), Core (Req), Code List, Described Value Domain, and Implements (USUBJID).

Figure 5. Data Standards Browser – Searching

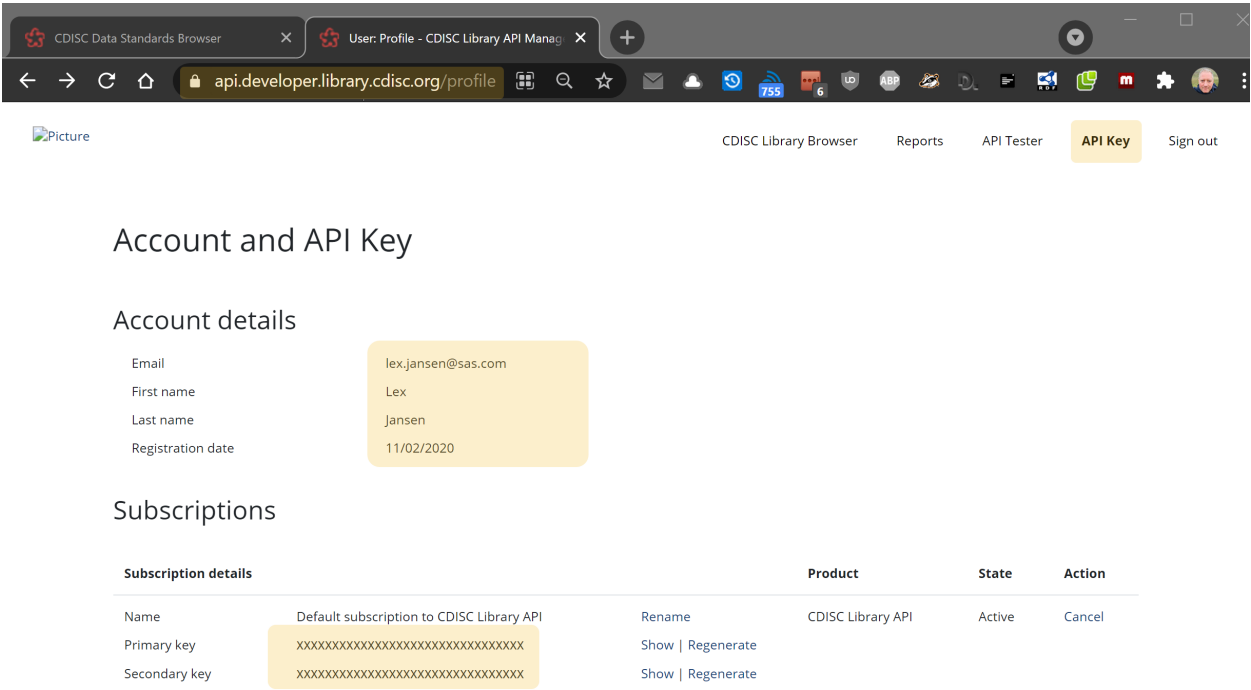
The screenshot shows the CDISC Library Data Standards Browser interface with a search bar at the top right containing the text 'vstest'. Below the search bar, a 'Close search results' button is visible. The main content area is titled 'Search Results' and shows '24 results for "vstest"'. A 'Narrow search results' section includes a 'Scope' dropdown set to 'Product' and a 'Scope Value' dropdown set to 'Select'. A list of search results is displayed, each starting with 'VSTEST' followed by its context (Data Collection Field, SDTM Dataset Variable, etc.) and a brief description. A dropdown menu is open next to the 'Scope Value' dropdown, showing a list of CDISC standards with checkboxes: CDASHIG v2.0, CDASHIG v2.1, SDTMIG v3.1.2, SDTMIG v3.1.3, SDTMIG v3.2, SDTMIG v3.3, SENDIG v3.0, SENDIG v3.1, SENDIG v3.1.1, and CDASHIG v1.1. A 'Reset' button is located to the right of the 'Scope Value' dropdown.

CDISC Library REST API

The CDISC Library API is REST-based and by default returns JSON content. Before you can use the CDISC Library API you will need to create API keys at the API Management Developer Portal:

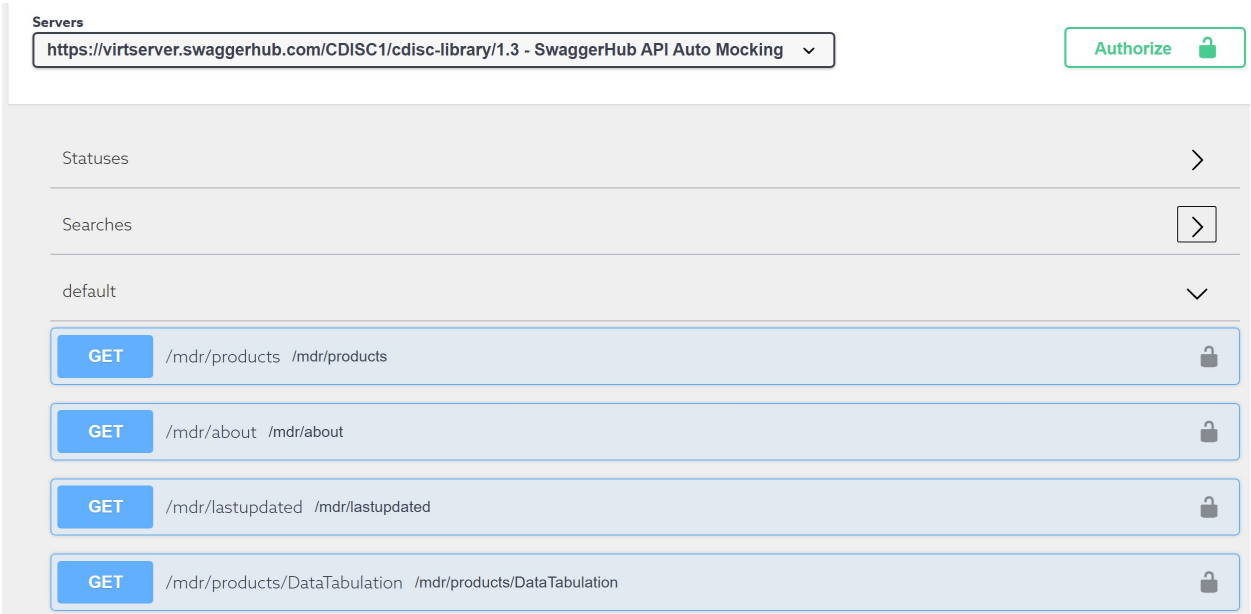
<https://api.developer.library.cdisc.org>

Figure 6. CDISC Library API Management Developer Portal



The CDISC Library API documentation describes how to request each API endpoint and what to expect as a response: <https://www.cdisc.org/cdisc-library/api-documentation>

Figure 7. CDISC Library REST API Documentation



CDISC Library REST API - Media Types

The CDISC Library API supports various media types, which are the supported formats for the responses that the API returns. Every API call supports the JSON format, which is the most common format in REST APIs nowadays. Formats like CSV and Excel are only returned when requesting complete Data Standards (E.g. SDTM-IG 3.3, SDTM 1.7) or complete Controlled Terminology packages (E.g. ADaM 2020-11-06).

Figure 8. CDISC Library REST API - Media Types

API category (highlighted rows) or specifics	JSON application/json	XML application/xml	CSV text/csv	Excel application/vnd.ms-excel
Statuses	✓			
Searches	✓			
default <small>① Referring to the category in the API Documentation</small>	✓	✓		
Controlled Terminology (CT)				
• /mdr/ct/packages/{product}	✓	✓	✓	✓
• All other CT endpoints	✓	✓		
Clinical Data Acquisition Standards Harmonization (CDASH)				
• /mdr/cdash/{version}	✓	✓	✓	✓
• All other CDASH endpoints	✓	✓		
CDASH Implementation Guide (CDASHIG)				
• /mdr/cdashig/{version}	✓	✓	✓	✓
• All other CDASHIG endpoints	✓	✓		
Study Data Tabulation Model (SDTM)				
• /mdr/sdtm/{version}	✓	✓	✓	✓
• All other SDTM endpoints	✓	✓		
SDTM Implementation Guide (SDTMIG)				
• /mdr/sdtmig/{version}	✓	✓	✓	✓
• All other SDTMIG endpoints	✓	✓		

REST API

WHAT IS AN API

API (Application Programming Interface) simplifies programming by abstracting the underlying implementation and only exposing objects or actions the developer needs. An API defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc. Through information hiding, APIs enable modular programming, allowing users to use the interface independently of the implementation. [2]

REST

REST (REpresentational State Transfer) is an architectural style for providing data exchange standards between computer systems on the web - over HyperText Transfer Protocol (HTTP) - making it easier for systems to communicate with each other [3].

RESTful systems, are characterized by how they use a uniform and predefined set of stateless operations and separate the concerns of client and server.

First presented by Roy Fielding in 2000 [4] in a famous dissertation REST has become the standard for interacting with independent systems across the web.

Figure 9. REST



REST Guiding Principles

Six guiding constraints define a RESTful system. These constraints restrict the ways that the server can process and respond to client requests so that, by operating within these constraints, the system gains desirable non-functional properties, such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability. If a system violates any of the required constraints, it cannot be considered RESTful [3].

The formal REST constraints are as follows:

- Client-Server – separation of concerns
- Stateless Server - Each request from client to server must contain all of the information necessary to understand the request
- Cacheable - If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests. This can improve efficiency.
- Uniform Interface - REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state (HATEOAS).
- Layered System - components are grouped, i.e., layered, in a hierarchical arrangement
- Code on Demand - REST allows client functionality to be extended on the client

REST Anatomy

REST uses a resource identifier (URIs) to identify particular resources

Figure 10. The Anatomy of a REST URI



The GET method is used for retrieving the information

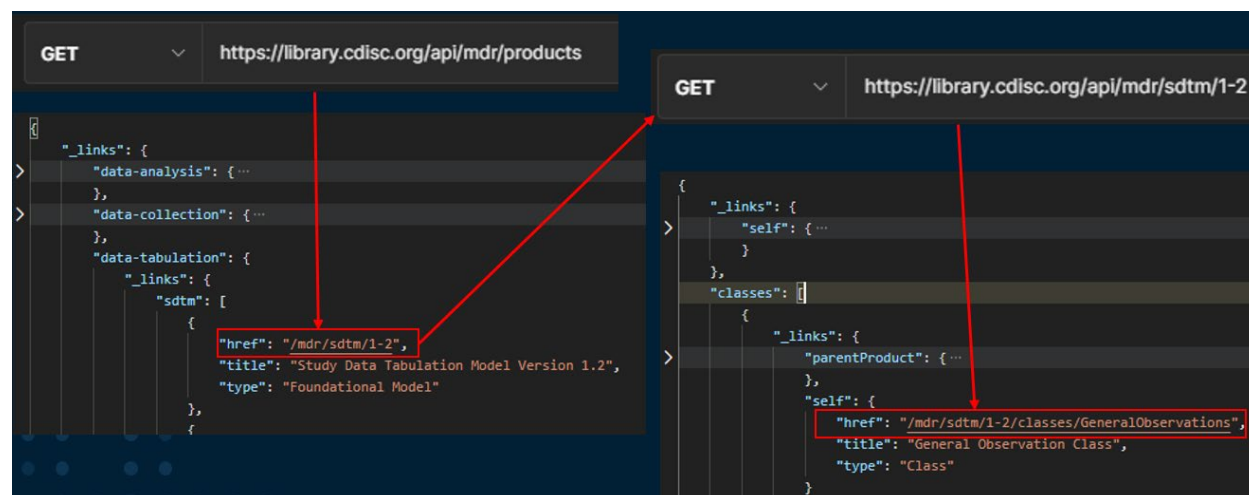
HATEOAS

HATEOAS (Hypermedia as the Engine of Application State) is a constraint of REST that distinguishes it from other network applications [5].

With HATEOAS a REST client enters a REST application through a simple fixed URL. All future actions the client may take are discovered within resource representations returned from the server. Thus, the response message returned from the first request contains links to create further requests.

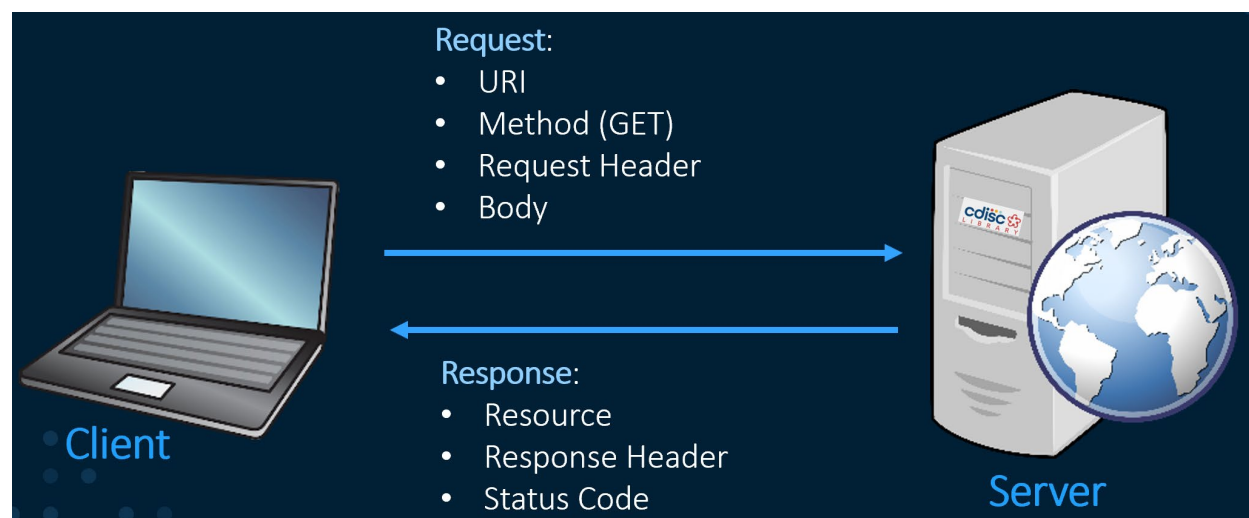
Figure 11 shows an example. The first request for the products gets a response in JSON that contains links to the individual products. A subsequent request for a product (SDTM 1.2) then gives a response with a link to, for example, the general observation classes in this specific product.

Figure 11. HATEOAS in REST



Request and Response

Figure 12. Request and Response Headers



Headers define the operating parameters of an HTTP transaction. Header fields are colon-separated key-value pairs in text format.

Request examples: Accept: application/json Accept-Charset: utf-8 Accept-Language: en-US Host: library.cdisc.org User-Agent: SAS/9 Connection: Keep-Alive	Response examples: Status: 200 OK Date: Fri, 16 Apr 2021 02:04:04 GMT Content-Type: application/json Content-Length: 300 Connection: Keep-Alive
---	---

Status Codes

Status codes are issued by a server in response to a client's request made to the server. [6]

- 1xx informational response – the request was received, continuing process
- 2xx successful – the request was successfully received, understood, and accepted
Example: 200 OK
- 3xx redirection – further action needs to be taken in order to complete the request
Example: 301 Moved Permanently
- 4xx client error – the request contains bad syntax or cannot be fulfilled
Examples: 400 Bad Request, 401 Unauthorized, 404 Not Found, 408 Request Timeout
- 5xx server error – the server failed to fulfil an apparently valid request
Example: 500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable

It is important to check the response code and determine further action based on the value.

HTTP REQUESTS IN SAS

PROC HTTP IN SAS

PROC HTTP is a powerful SAS procedure to issues Hypertext Transfer Protocol (HTTP) requests [7] [8] [9]. The procedure includes a DEBUG statement, response status macro variables, and the ability to specify a time-out period for requests.

Below are examples of requesting the CDISC Library Products with requests in JSON, XML and Microsoft Excel (XLSX).

It is expected that the macro variable &ApiKey contains your personal API key that you created at the CDISC Library API Management Developer Portal.

JSON Response

```

filename response temp;
proc http
  url="https://api.library.cdisc.org/api/mdr/products"
  out=response;
  headers
    "api-key" = "&ApiKey"
    "Accept" = "application/json";
run;

%put %sysfunc(jsonpp(response, log));

filename map temp;
libname response json map=map automap=create fileref=response;

proc copy in=response out=work;
run;

```

NOTE: PROCEDURE HTTP used (Total process time):

real time	0.50 seconds
cpu time	0.04 seconds

NOTE: 200 OK

```

{
  "_links": {
    "data-analysis": {
      "_links": {
        "adam": [
          {
            "href": "/mdr/adam/adam-2-1",
            "title": "Analysis Data Model Version 2.1",
            "type": "Foundational Model"
          },
          {
            "href": "/mdr/adam/adam-adae-1-0",
            "title": "Analysis Data Model Data Structure for Adverse Event Analysis Version 1.0",
            "type": "Implementation Guide"
          },
          {
            "href": "/mdr/adam/adam-occds-1-0",
            "title": "ADaM Structure for Occurrence Data (OCODS) Version 1.0",
            "type": "Implementation Guide"
          }
        ]
      }
    }
  }
}

```

26 filename map temp;

27 libname response json map=map automap=create fileref=response;

NOTE: JSON data is only read once. To read the JSON again, reassign the JSON LIBNAME.

NOTE: Map file C:\Users\frjans\AppData\Local\Temp\SAS Temporary Files\TD30412_110e010_\#LN00048 was created.

NOTE: Libref RESPONSE was successfully assigned as follows:

Engine:	JSON
Physical Name:	C:\Users\frjans\AppData\Local\Temp\SAS Temporary Files\TD30412_110e010_\#LN00047

28

29 proc copy in=response out=work;

30 run;

NOTE: Copying RESPONSE.ALLDATA to WORK.ALLDATA (memtype=DATA).

NOTE: BUFSIZE is not cloned when copying across different engines.

System Option for BUFSIZE was used.

















NOTE: There were 611 observations read from the data set RESPONSE.ALLDATA.

NOTE: The data set WORK.ALLDATA has 611 observations and 8 variables.

NOTE: Copying RESPONSE._LINKS_ADAM to WORK._LINKS_ADAM (memtype=DATA).

NOTE: BUFSIZE is not cloned when copying across different engines.

This would result in 16 data sets, some of which need to be merged to get the expected SAS data set.

 Alldata	 _links_adam	 _links_adam2	 _links_cdash
 _links_cdashig	 _links_packages	 _links_sdtm	 _links_sdtmig
 _links_sdtmig2	 _links_self	 _links_self2	 _links_self3
 _links_self4	 _links_self5	 _links_self6	 _links_sendig

XML Response

```
filename response temp;
proc http
  url="https://api.library.cdisc.org/api/mdr/products"
  out=response;
  headers
    "api-key" = "&ApiKey"
    "Accept" = "application/xml";
run;

filename map temp;
libname response xmlv2 xmlmap=map automap=replace fileref=response;

proc copy in=response out=work;
run;
```

NOTE: PROCEDURE HTTP used (Total process time):

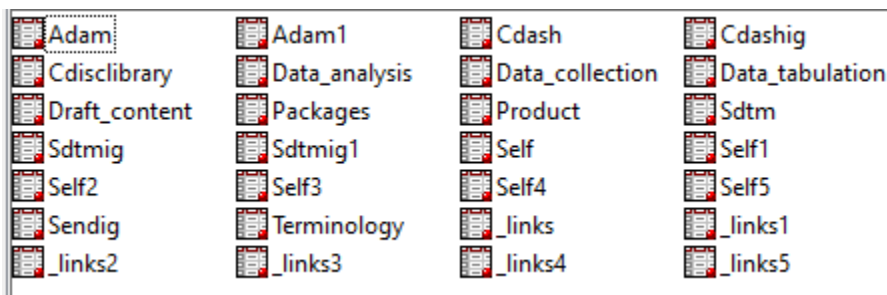
```
real time      0.45 seconds
cpu time       0.00 seconds
```

NOTE: 200 OK

```
14 filename map temp;
15 libname response xmlv2 xmlmap=map automap=replace fileref=response;
NOTE: Processing XMLMap version 2.1.
NOTE: Libref RESPONSE was successfully assigned as follows:
      Engine:          XMLV2
      Physical Name:   C:\Users\LEXJAN~1\AppData\Local\Temp\SAS Temporary Files\_TD21964_WIN10PRO64\_LN00012
16 proc copy in=response out=work;
NOTE: Writing HTML Body file: sashtml.htm
17 run;

NOTE: Copying RESPONSE._links to WORK._LINKS (mimetype=DATA).
NOTE: BUFSIZE is not cloned when copying across different engines. System Option for BUFSIZE was used.
INFO: Data set block I/O cannot be used because:
INFO:   - The data sets use different engines, have different variables or have attributes that may differ.
NOTE: There were 1 observations read from the data set RESPONSE._links.
NOTE: The data set WORK._LINKS has 1 observations and 2 variables.
NOTE: Compressing data set WORK._LINKS decreased size by 0.00 percent.
      Compressed is 1 pages; un-compressed would require 1 pages.
NOTE: Copying RESPONSE._links1 to WORK._LINKS1 (mimetype=DATA).
NOTE: BUFSIZE is not cloned when copying across different engines. System Option for BUFSIZE was used.
INFO: Data set block I/O cannot be used because:
INFO:   - The data sets use different engines, have different variables or have attributes that may differ.
NOTE: There were 1 observations read from the data set RESPONSE._links1.
NOTE: The data set WORK._LINKS1 has 1 observations and 2 variables.
NOTE: Compressing data set WORK._LINKS1 decreased size by 0.00 percent.
      Compressed is 1 pages; un-compressed would require 1 pages.
NOTE: Copying RESPONSE._links2 to WORK._LINKS2 (mimetype=DATA).
```

This would result in 28 data sets, some of which need to be merged to get the expected SAS data set.



Excel Spreadsheet response

```
filename response "C:/_Projects/CDISC Library/response_xls/SDTMIG_3.3.xlsx";  
proc http  
  url="https://library.cdisc.org/api/mdr/sdtmig/3-3"  
  out=response;  
  headers  
    "api-key"="&ApiKey"  
    "Accept"="application/vnd.ms-excel";  
run;  
  
proc import datafile=response  
  out=work.sdtmig_3_3_tables dbms=xlsx replace;  
  sheet="Datasets";  
run;  
  
proc import datafile=response  
  out=work.sdtmig_3_3_columns dbms=xlsx replace;  
  sheet="Variables";  
run;
```

Beginning with SAS 9.4M5, PROC HTTP sets up macro variables after execution:

SYS_PROCHTTP_STATUS_CODE

stores the status code of the HTTP request

SYS_PROCHTTP_STATUS_PHRASE

stores the descriptive phrase that is associated with the status code

These macro variables can be used to determine the action based on the value of the status code in the response. An example of the use of these macro variables is in the *prochttp_check_return* macro below.

```
%macro prochttp_check_return(code);  
  %if %symexist(SYS_PROCHTTP_STATUS_CODE) ne 1 %then %do;  
    %put ERROR: Expected &code., but no response received from the HTTP Procedure;  
    %abort;  
  %end;  
  %else %do;  
    %if &SYS_PROCHTTP_STATUS_CODE. ne &code. %then %do;  
      %put ERROR: Expected &code.,  
        but received &SYS_PROCHTTP_STATUS_CODE &SYS_PROCHTTP_STATUS_PHRASE;  
      %abort;  
    %end;  
  %end;  
%mend prochttp_check_return;
```

As we saw, the JSON and XML responses can result in a large number of SAS data sets that need to be merged. The table below shows some numbers for specific endpoints.

endpoint	# SAS data sets (JSON)	# SAS data sets (XML)
<i>mdr/products</i>	16	18
<i>mdr/ct/packages/sdtmct-2021-03-26</i>	8	7
<i>mdr/sdtmig/3-3</i>	28	32
<i>mdr/sdtm/1-7</i>	39	46

The need to merge many SAS data sets can be avoided by requesting the response in Excel (XLSX) format. However, this would limit the SAS programmer to requests for full data Standards and Control Terminology Packages.

Also, nowadays the default response format when using a REST API as the data exchange standard between computer systems on the web is JSON.

This means that it is important to have a good framework for parsing REST API responses in JSON when using SAS.

Although one can use a JSON Map there are several challenges to this approach. Many of these challenges are the same as with XML [10].

The rest of this paper describes how SAS with PROC LUA provides for an efficient way to manage and interface with a REST API and to parse JSON responses.

All code used in this paper is available at GitHub: <https://github.com/lexjansen/sas-papers>

USING PROC LUA IN SAS TO MANAGE HTTP REQUESTS

Base SAS® 9.4 introduced the LUA procedure which embeds Lua 5.2. Lua was created in 1993 by Roberto Ierusalimsky (Pontifical Catholic University of Rio de Janeiro) [11][12].

Lua is a modern programming language with a very simple syntax and supports highly flexible data structures and modules. Tables are the only data structure available in Lua which can be used to create different types like arrays and dictionaries. Lua is known for excellent performance, both in speed and memory.

Lua does not replace the SAS DATA step or procedures but enhances the ability to drive SAS. PROC LUA has direct access to the vast majority of SAS functions

Lua makes parsing JSON very easy with one of the available modules for encoding and decoding JSON.

PROC LUA runs the Lua virtual machine inside the SAS process.

Lua code is executed within a SUBMIT / ENDSUBMIT block in PROC LUA.

Lua is a dynamically typed language – variables do not have types; only values do

Basic types in Lua are:

nil, boolean, number, string, function, userdata, thread, and table

```
proc lua ;
  submit;

  -- Lua statements in SAS
  print('Hello world')

endsubmit;
run;
```

Tables are the sole data-structuring mechanism in Lua

They can be used to represent ordinary arrays, associative arrays, lists, symbol tables, sets, records, graphs, trees, etc.

Some examples of the use of Lua tables are below.

```
-- create a table as an array
products = {'sdtm', 'sdtmig', 'sendig'}

local product = products[1]

-- loop over the array
for i, product in ipairs(products) do
  print(i, product)
end
```

```
-- create a table as a hash table of tables
local extract_attributes = {
  model = {
    template_tables = 'tplts.tabulation_columnclassgroup',
    template_columns = 'tplts.tabulation_columnclass'},
  ig = {
    template_tables = 'tplts.tabulation_table',
    template_columns = 'tplts.tabulation_column'}
}

local model_tables = extract_attributes.model.template_tables
-- or
local ig_columns = extract_attributes['ig']['template_columns']
```

```
-- loop over the tables
for i, table in pairs(extract_attributes) do
  for key, attribute in pairs(table) do
    print(i, key, attribute)
  end
end
```

PROC LUA creates a special global Lua table called **sas**. The **sas** table contains functions: **sas.scan**, **sas.symget**, **sas.symput**, ...

```
%let foo=conference;

proc lua;
  submit;
    local foo = sas.symget("foo")
    print("foo is ", foo) -- prints 'conference'
    sas.symput('foo','pharmasug')
  endsubmit;
run;

%put &foo; /* prints 'pharmasug' */
```

PROC LUA can also submit SAS code with the **sas.submit** function. An optional table parameter with key-value pairs can be made available for resolution in the block of SAS code.

```
local products_dataset = "prod.products"

sas.submit([[
  proc sort data=@dataset@;
    by @sort_key@;
  run;
]], { dataset=products_dataset, sort_key="product_href" })
```

```
proc sort data=prod.products;
  by product_href;
run;
```

PROC LUA can read SAS datasets by opening the SAS data set with the **sas.open** function and using the **sas.get_value** function to get observation values.

For example, if we have the following data set:

productclass_title	productclass_href	product_type	product_title	product_href
Product Group Data Collection	/mdr/products/DataCollection	Foundational Model	Clinical Data Acquisition Standards Harmonization Model Version 1.1	/mdr/cdash/1-1
Product Group Data Collection	/mdr/products/DataCollection	Implementation Guide	Clinical Data Acquisition Standards Harmonization (CDASH) User Guid...	/mdr/cdashig/1-1-1
Product Group Data Collection	/mdr/products/DataCollection	Implementation Guide	Clinical Data Acquisition Standards Harmonization Implementation Guid...	/mdr/cdashig/2-0
Product Group Data Collection	/mdr/products/DataCollection	Implementation Guide	Clinical Data Acquisition Standards Harmonization Implementation Guid...	/mdr/cdashig/2-1
Product Group Terminology	/mdr/products/Terminology	Terminology	ADaM Controlled Terminology Package 19 Effective 2014-09-26	/mdr/ct/packages/adamct-2014-09-26
Product Group Terminology	/mdr/products/Terminology	Terminology	ADaM Controlled Terminology Package 24 Effective 2015-12-10	/mdr/ct/packages/adamct-2015-12-10

the following code shows how to read this data set in Lua.

```

local dsid_prod = sas.open("prod.products")
sas.where(dsid_prod, "productclass_title = 'Product Group Terminology'") -- Only Terminology

-- loop over all products
while sas.next(dsid_prod) do

    product_type = sas.get_value(dsid_prod, 'product_type')
    product_href = sas.get_value(dsid_prod, 'product_href')
    product = sas.scan(product_href, -1, '/')
    pubdate = string.gsub(string.sub(product, string.len(product)-9), '-', '')
    ctproduct = string.sub(product, 1, string.len(product)-11)

    ...

end -- end of products loop
sas.close(dsid_prod)

```

PROC LUA can also write to SAS datasets. Use the **sas.new_table** function to create a new (empty) data set template. The following Lua function (cdisclibrary.products) creates a 0-observation SAS data set:

```

function cdisclibrary.products (dataset_name)

    local dsid

    sas.new_table(dataset_name, {
        {name="productclass_title", type="C", length=200, label="Product Class Title"},
        {name="productclass_href", type="C", length=100, label="Product Class Link"},
        {name="product_type", type="C", length=100, label="Product Type"},
        {name="product_title", type="C", length=200, label="Product Title"},
        {name="product_href", type="C", length=64, label="Product Link"}
    })
    dsid = sas.open(dataset_name, "u")

    return dsid
end

```

A typical scenario would be to open a new 0-observation data set and write out observations in some sort of loop over a Lua table by using the **sas.put_value** function:

```

local dsid = cdisclibrary.products("prod.products")

```



```

for some-condition do

    local productclass_href = ...
    local product_type = ...
    local product_title = ...
    local product_href = ...

    sas.append(dsid)

    sas.put_value(dsid, "productclass_href", productclass_href)
    sas.put_value(dsid, "product_type", product_type)
    sas.put_value(dsid, "product_title", product_title)
    sas.put_value(dsid, "product_href", product_href)

    sas.update(dsid)
end

sas.close(dsid)

```

PARSING THE JSON RESPONSES WITH A LUA JSON LIBRARY

The code at GitHub (<https://github.com/lexjansen/sas-papers>) contains several Lua modules to support extraction from the CDISC Library:

rest.lua	a Lua module to easily interface with a REST API from SAS [13]
json.lua	a Lua module for simple JSON encoding and decoding in pure Lua [14]
cdisclibrary.lua	a Lua module for converting CDISC Library JSON responses into SAS data sets
fileutils.lua,	
utils.lua,	
stringutils.lua	Lua modules with support functions

rest.lua - The heart of this module is a function: **rest.request**

```

----- Submit a request to rest.base_url
-- ARGUMENTS:
-- action      - string - 'GET', 'POST', 'PUT', etc.
-- request     - string - the portion of the URL that follows rest.base_url
-- body_out    - string - [OPTIONAL] the fileref to write the output to. Defaults to '_bout_'
-- content_type - string - [OPTIONAL] passed to PROC HTTP's 'ct' parameter (content type)
-- header_out  - string - [OPTIONAL] the fileref to write the returned header to. Defaults to '_hout_'
-- body_in     - string - [OPTIONAL] fileref for the request body.
-- header_in   - string - [OPTIONAL] fileref for the request header.
-- RETURNS:
-- pass        - boolean - whether or not the HTTP return code was in the 200's (200 OK, 201 CREATED, etc)
-- code        - number  - the actual http return code
function rest.request( action, request, body_out, content_type, header_out, body_in, header_in)
    ...
    return pass, code
end

```

An example of using the `rest.request` function is to get information from the CDISC Library about the dates when products were last updated and create a Lua table from that information. We can use this table then to determine whether we should do a new REST API request.

```

filename luapath ("../lua");
filename lastupd "_lastupdated_.json";

proc lua restart;
  submit;

  rest = require 'rest'
  json = require 'json'

  local token = ''
  rest.base_url = 'https://library.cdisc.org/api'
  rest.headers="Accept="application/json" api-key="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"

  local pass,code = rest.request('get','mdr/lastupdated', 'lastupd')
  local lastupdated = json:decode(rest.utils.read('lastupd'))

  local lastupdated_table={}
  for key, value in pairs(lastupdated) do
    if key ~= "_links" then lastupdated_table[key] = value end
  end

  endsubmit;
run;

```

The JSON response look like this:

```

{
  "_links": {
    "self": {
      "href": "/mdr/lastupdated",
      "title": "Last Update Date of CDISC Library Products",
      "type": "About CDISC Library"
    }
  },
  "data-analysis": "2020-11-11",
  "data-collection": "2020-11-11",
  "data-tabulation": "2021-03-30",
  "draft-content": "2021-03-30",
  "overall": "2021-03-30",
  "terminology": "2021-03-30"
}

```

The resulting Lua table:

```

{
  ["data-collection"]="2020-11-11"
  ["overall"]="2021-03-30"
  ["data-analysis"]="2020-11-11"
  ["terminology"]="2021-03-30"
  ["draft-content"]="2021-03-30"
  ["data-tabulation"]="2021-03-30"
}

```

We can use the resulting Lua table to determine whether we should do a new extraction:

```

local response_folder = sas.symget("response_folder")
local response_file = sas.io.join(response_folder, "_products.json")
sas.filename('products', response_file)

-- If the response file does not exist or is out-of-date then retrieve it
if (not sas.fileexists(response_file)) or
    (fileutils.lastmodified('products') < lastupdated['overall']) then
    local pass,code = rest.request('get','mdr/products','products')
end

-- convert the JSON response file into a Lua table
cdisc_products = json:decode(rest.utils.read('products'))._links

-- parse the Lua table and convert it to a SAS dataset
local dsid = cdisclibrary.products(products_dataset)
cdisclibrary.add_product_to_dataset (dsid, cdisc_products)
if dsid then sas.close(dsid) end

```

The **cdisclibrary.lua** module contains functions to create SAS data sets from JSON responses from CDISC Library requests for:

products catalog

Controlled Terminology packages

Tabulation products:

- Foundational Models (sdtm)
- Implementation Guides (sdtmig, sendig)

All code can be found at GitHub: <https://github.com/lexjansen/sas-papers>

Figures 13-16 show some SAS data sets that are created by the code.

Figure 13. SDTM 1.7 Class Tables

Library: SDTM_1_7_CLASS_TABLES SDTM_1_7_CLASS_COLUMNS										
Table View										
	name	label	description	effectiveDate	registration	version	class_name	class_label	class_description	class_structure
1	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	General Observations	General Observat...	The majority of observations ...	
2	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Interventions	Interventions Obs...	This SDTM class captures in...	
3	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Events	Events Observati...	This SDTM class captures pl...	
4	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Findings	Findings Observa...	This SDTM class captures th...	
5	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Findings About	Findings About E...	This SDTM class is a speciali...	
6	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Special-Purpose	Special-Purpose ...	This SDTM class contains a ...	
7	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Trial Design	Trial Design Model	This SDTM class describes t...	
8	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Relationship	Relationship Dat...	This SDTM class provides a ...	
9	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Study Reference	Datasets for Stud...	This special purpose SDTM ...	
10	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Associated Persons	Associated Perso...	Associated Persons (AP) are ...	
11	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Special-Purpose	Demographics	A special-purpose domain th...	DM
12	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Special-Purpose	Comments	A special-purpose domain th...	CO
13	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Special-Purpose	Subject Element	A special-purpose domain th...	One Record per Actual ... SE
14	SDTM v1.7	Study Data Tabul...	This document d...	2018-11-20	Final	1.7	Special-Purpose	Subject Meta	A special-purpose domain th...	One Record per Subject ... SE

Figure 14. SDTM 1.7 Class Columns

Library: SDTM_1_7_CLASS_TABLES SDTM_1_7_CLASS_COLUMNS												
Freeze Hide Show... Format Filter... Font... Find												
Table View												
	name	label	effectiveDate	version	class_name	dataset_name	variable_ordinal	variable_name	variable_label	variable_description	variable_simplified	variable_role
1	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		1	STUDYID	Study Identifier	Unique identifier f...	Char	Identifier
2	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		2	DOMAIN	Domain Abbrevia...	2-character abbr...	Char	Identifier
3	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		3	USUBJID	Unique Subject I...	Identifier used to ...	Char	Identifier
4	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		4	APID	Associated Perso...	Identifier for a sin...	Char	Identifier
5	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		5	POOLID	Pool Identifier	An identifier used...	Char	Identifier
6	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		6	SPDEVID	Sponsor Device I...	Sponsor-defined i...	Char	Identifier
7	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		7	NHOID	Non-Host Organi...	Sponsor-defined i...	Char	Identifier
8	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		8	FETUSID	Fetus Identifier	Identifier used to i...	Char	Identifier
9	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		9	FOCID	Focus of Study-S...	Identification of a...	Char	Identifier
10	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		10	-SEQ	Sequence Number	Sequence numb...	Num	Identifier
11	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		11	-GRPID	Group ID	Optional group id...	Char	Identifier
12	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		12	-REFID	Reference ID	Optional internal ...	Char	Identifier
13	SDTM v1.7	Study Data Tabulation Mo...	2018-11-20	1.7	General Observations		13	-RECID	Invariant Record ...	Identifier for a rec...	Char	Identifier

Figure 15. SDTM-IG 3.3 Tables

Library: SDTMIG_3_3_TABLES SDTMIG_3_3_COLUMNS										
Freeze Hide Show... Format Filter... Font... Find										
Table View										
	name	label	effectiveDate	version	class_name	table_ordinal	table_name	table_label	table_description	table_structure
1	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Special-Purpose	1	CO	Comments	A special-purpose domain...	One record per comment per subject
2	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Special-Purpose	2	DM	Demographics	A special-purpose domain...	One record per subject
3	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Special-Purpose	3	SE	Subject Elements	A special-purpose domain...	One record per actual Element per s...
4	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Special-Purpose	4	SM	Subject Disease ...	A special-purpose domain...	One record per Disease Milestone p...
5	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Special-Purpose	5	SV	Subject Visits	A special-purpose domain...	One record per subject per actual visit
6	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Interventions	6	AG	Procedure Agents	An interventions domain t...	One record per recorded interventio...
7	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Interventions	7	CM	Concomitant/Prio...	An interventions domain t...	One record per recorded interventio...
8	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Interventions	8	EX	Exposure	An interventions domain t...	One record per protocol-specified st...
9	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Interventions	9	EC	Exposure as Coll...	An interventions domain t...	One record per protocol-specified st...
10	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Interventions	10	ML	Meal Data	An interventions domain t...	One record per food product occur...
11	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Interventions	11	PR	Procedures	An interventions domain t...	One record per recorded procedure ...
12	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Interventions	12	SU	Substance Use	An interventions domain t...	One record per substance type per r...
13	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Events	13	AE	Adverse Events	An events domain that co...	One record per adverse event per s...
14	SDTMIG v3.3	Study Data Tabul...	2018-11-20	3.3	Events	14	CE	Clinical Events	An events domain that co...	One record per event per subject

Figure 16. SDTM-IG 3.3 Columns

Library: SDTMIG_3_3_TABLES SDTMIG_3_3_COLUMNS													
Freeze Hide Show... Format Filter... Font... Find													
Table View													
	name	label	effectiveDate	table_name	column_name	column_label	column_description	column_ordinal	column_role	column_simpleData	column_core	column_codelist	column_valuelist
1	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	STUDYID	Study Identifier	Unique identifier for a study.	1	Identifier	Char	Req		
2	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	DOMAIN	Domain Abbreviation	Two-character abbreviation...	2	Identifier	Char	Req		CO
3	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	RDOMAIN	Related Domain Abbrevi...	Two-character abbreviation...	3	Record Qualifier	Char	Pem		
4	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	USUBJID	Unique Subject Identifier	Identifier used to uniquely i...	4	Identifier	Char	Req		
5	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	COSEQ	Sequence Number	Sequence Number given to ...	5	Identifier	Num	Req		
6	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	IDVAR	Identifying Variable	Identifying variable in the p...	6	Record Qualifier	Char	Pem		
7	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	IDVARVAL	Identifying Variable Value	Value of identifying variable ...	7	Record Qualifier	Char	Pem		
8	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	COREF	Comment Reference	Sponsor-defined reference ...	8	Record Qualifier	Char	Pem		
9	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	COVAL	Comment	The text of the comment. T...	9	Topic	Char	Req		
10	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	COEVAL	Evaluator	Used to describe the origin...	10	Record Qualifier	Char	Pem		
11	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	COEVALID	Evaluator Identifier	Used to distinguish multiple ...	11	Record Qualifier	Char	Pem	C96777	
12	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	CODTC	Date/Time of Comment	Date/time of comment on d...	12	Timing	Char	Pem		ISO 8601
13	SDTMIG v3.3	Study Data Tabul...	2018-11-20	CO	CODY	Study Day of Comment	Study day of the comment. ...	13	Timing	Num	Pem		
14	SDTMIG v3.3	Study Data Tabul...	2018-11-20	DM	STUDYID	Study Identifier	Unique identifier for a study.	1	Identifier	Char	Req		
15	SDTMIG v3.3	Study Data Tabul...	2018-11-20	DM	DOMAIN	Domain Abbreviation	Two-character abbreviation...	2	Identifier	Char	Req		DM

LOADING DATA STANDARDS INTO SAS LIFE SCIENCE ANALYTICS FRAMEWORK

The last part of the paper focusses on loading data sets extracted from the CDISC Library into the Data Standards and Controlled Terminology areas of the Clinical Management module of SAS® Life Science Analytics Framework (LSAF) [15].

Standards metadata and Controlled Terminology metadata can be loaded into SAS® Life Science Analytics Framework (LSAF). Before this can happen some model customizations in LSAF may need to take place:

- To support newer versions
- To support Tabulation Class Structures and Analysis Data Structures

Some issues in the metadata may need to be resolved, for example:

- Missing metadata
example: in SDTM-IG 3.2 the required LSAF submissiondatatype is missing for TR.TRMETHOD
- Various SDTM-IG variables have multiple codelist references
example: in SDTM-IG 3.3 the DS.DSDECOD variable has C114118 and C66727

Variable names need to be mapped to the names that LSAF requires. The example in GitHub (<https://github.com/lexjansen/sas-papers>) has a simple Excel based mapping framework for this.

	A	B	C	D	
1	TABLE	SOURCE	TARGET	ACTION	VALUE
80	CODELIST	EFFECTIVEDATE	RELEASEDATE	RENAME	
81	CODELIST		SOURCE	ASSIGN	"NCI Thesaurus"
82	CODELIST	VERSION	SOURCEVERSION	RENAME	
83	CODELIST	REGISTRATIONSTATUS		DROP	
84	CODELIST	HREF		DROP	
85	CODELIST	CODELIST_SUBMISSIONVALUE	CODELIST_SHORTNAME	RENAME	
86	CODELIST	CODELIST_NAME	CODELIST_NAME	RENAME	
87	CODELIST	CODELIST_DEFINITION	CODELIST_DESCRIPTION	RENAME	
88	CODELIST	CODELIST_CONCEPTID	CODELIST_CODE	RENAME	
89	CODELIST		CODELIST_DATATYPE	ASSIGN	"text"
90	CODELIST		CODELIST_SASFORMATNAME	ASSIGN	
91	CODELIST	CODELIST_PREFERRED_TERM	CODELIST_PREFERRED_TERM	RENAME	
92	CODELIST	CODELIST_SYNONYMS		DROP	
93	CODELIST	CODELIST_EXTENSIBLE	CODELIST_EXTENSIBLE	RENAME	
94	CODELIST		CODELIST_EXTENSIBLE_STUDY	ASSIGN	CODELIST_EXTENSIBLE
95	CODELIST		CODELIST_SUBSETTABLE	ASSIGN	"Yes"

A macro can use this mapping table to convert extracted data sets to the required format.

```
%map_extract_to_lsaf(
    mappings=maps.mapping,
    tabletype=codelist,
    template=tplts.terminology,
    source=extract.sdtmct_20200327,
    target=ct.sdtmct_20200327
);
```

Figure 17. Data Standards in SAS® Life Science Analytics Framework

SAS® Life Science Analytics Framework - Data Standards							
<div> <div>Published</div> <div>In progress</div> <div>Checked out</div> </div> <div> <input type="text" value="Text to filter"/> 7 standards </div>							
<input type="checkbox"/>	S..	S ↑	Name	Type	Standard Model	Base S...	Base Stande
<input type="checkbox"/>		0	CDISC ADaM 2.1	Analysis	Analysis Standard	ADaM	1.1
<input type="checkbox"/>		0	CDISC SDTM 3.2	Tabulation	Tabulation Standard	SDTM	3.2
<input type="checkbox"/>		0	CDISC SDTM 3.2 with...	Tabulation	Tabulation Standa...	SDTM	3.2
<input type="checkbox"/>		0	CDISC SEND 3.1	Tabulation	Tabulation Standard	SEND	3.1
<input type="checkbox"/>		0	SDTM Dermatitis	Tabulation	Tabulation Standard	SDTM	3.2
<input type="checkbox"/>		0	SDTM Pneumococcal	Tabulation	Tabulation Standard	SDTM	3.2
<input type="checkbox"/>		0	SDTM Vaccines	Tabulation	Tabulation Standard	SDTM	3.2

CONCLUSION

- SAS fully supports extraction from the CDISC Library with PROC HTTP
- PROC LUA greatly enhances SAS/Base capabilities to manage REST API requests and to parse JSON files.

REFERENCES

- [1] CDISC Webinar April 2021: Ideas for using the CDISC Library and a Look at What's Coming Next (Anthony Chow, Sam Hume)
<https://www.cdisc.org/events/webinar/cdisc-library-ideas-using-cdisc-library-and-look-whats-coming-next>
- [2] Wikipedia, "API", <https://en.wikipedia.org/wiki/API>
- [3] Wikipedia, "Representational state transfer",
https://en.wikipedia.org/wiki/Representational_state_transfer
- [4] [Architectural Styles and the Design of Network-based Software Architectures](#)
Roy Thomas Fielding, 2000
- [5] Wikipedia, "HATEOAS", <https://en.wikipedia.org/wiki/HATEOAS>
- [6] Wikipedia, "List of HTTP status codes", https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
- [7] Base SAS Procedures Guide: [HTTP Procedure](#)
- [8] [REST Just Got Easy with SAS® and PROC HTTP](#)
Joseph Henry, SAS Global Forum 2020
- [9] [The ABCs of the HTTP Procedure](#)
Joseph Henry, SAS Global Forum 2019
- [10] [Accessing the Metadata from Define-XML](#)
Lex Jansen, PharmaSUG 2018

- [11] The Programming Language Lua (<http://www.lua.org>)
Lua.org (2013)
- [12] [Driving SAS® with Lua](#)
Paul Tomas, SAS Global Forum 2015
- [13] [REST Easier with SAS®: Using the LUA Procedure to Simplify REST API Interactions](#)
Steven Major, SAS Global Forum 2019
(code on [GitHub](#))
- [14] A Lua module for simple JSON encoding and decoding in pure Lua
Jeffrey Friedl, 2010-2017
<http://regex.info/blog/lua/json>
- [15] SAS® Life Science Analytics Framework,
https://www.sas.com/en_us/software/life-science-analytics-framework.html

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Lex Jansen
SAS Institute Inc.
Email: lex.jansen@sas.com



SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Lua software: Copyright © 2020–2021 Lua.org, PUC-Rio.