



Orbis Clinical
Life Sciences Solutions
an Epic Staffing Group company

How the R Programming Language Handles the CDISC Library API

Joseph Hinson

- What are APIs and REST APIs?
- What is CDISC Library?
- The 4-step strategy used in R to obtain info from CDISC Library:
 - Make a REST API call to CDISC Library API
 - Split response into string segments with each segment for an SDTM variable
 - Use Regular Expressions to extract variable metadata from each segment
 - Assemble all the extracted info from all the segments into a Data Frame





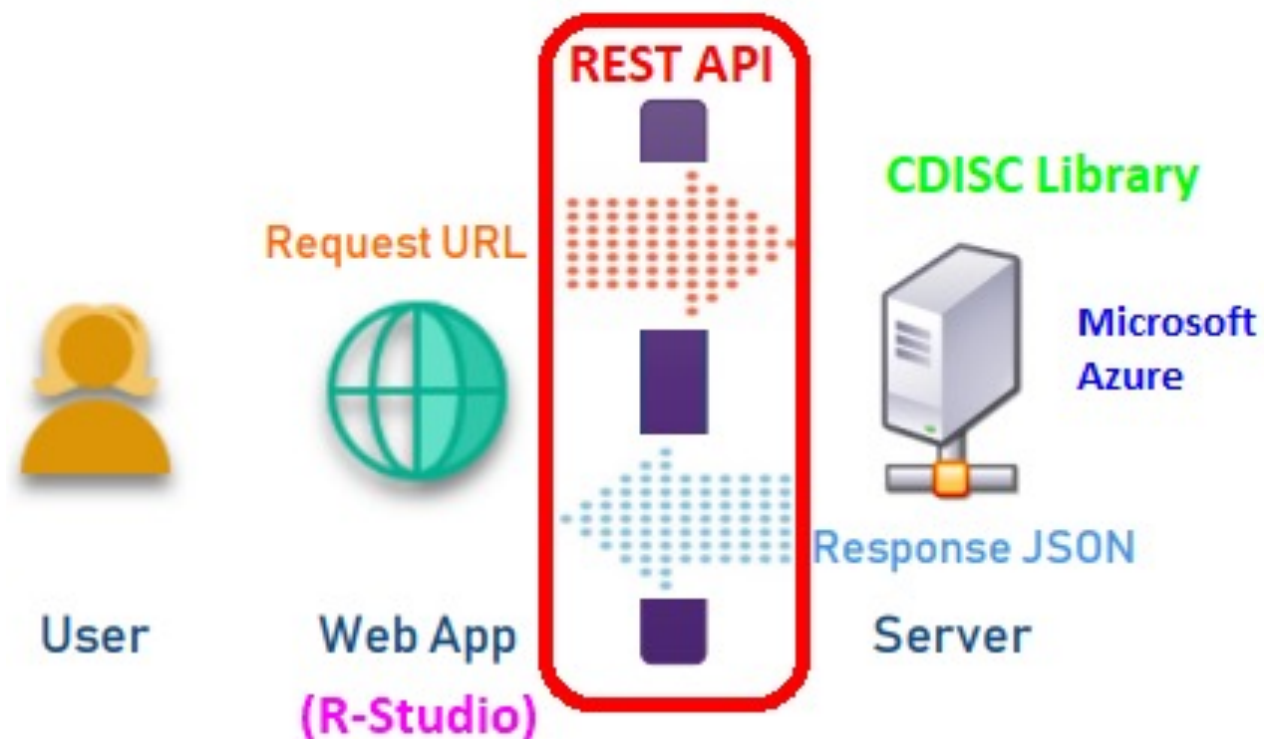
[Slide 3] What are Application Programming Interfaces (APIs) and REST APIs?

- The Web is no longer for fetching HTML pages only
- The Web can also be used by software applications to obtain data
- APIs allow communication between diverse software applications
- “REST” stands for “REpresentational State Transfer”
- REST API is a particular data exchange protocol, that allows the use of HTTP verbs (e.g. GET, POST) to request data on the web, getting a response in a standard format (JSON or XML).
- The use of REST APIs has grown exponentially
- CDISC Library API is a REST API



[Slide 4] What is CDISC Library?

- A huge metadata repository of CDISC standards
- Accessible via two interfaces:
 [Data Standards Browser](#) and [CDISC Library API](#)
- The two interfaces are based on linked-data and REST API



[Slide 5] Strategy-1: Making the REST API call

- The R package *httr* is used (*jsonlite* too can be used).
- The *httr* functions **GET ()**, **add_headers ()**, and **content ()** are involved
- An API-Key supplied by CDISC is required (a CDISC account is mandatory)
- The **GET ()** function requires two parameters:
 - Endpoint and Configuration Options
- The Endpoint is made up of the Primary URL and the CDISC sections of interest
- The Configuration Options are provided with the "**add_headers ()**" function.

Then, `Response <- GET (Endpoint, add_headers ("API-Key"=, "accept"=))`



[Slide 6] Issuing GET() to Obtain AE Metadata of SDTMIG 3.2 from CDISC Library

```
myAPIkey <- "12345abcde67890fghij"
PrimaryURL<- "https://library.cdisc.org/api/mdr/sdtmig/"
SDTMversion <- "3-2"
DOMAIN <- "AE"
endpoint <- paste0 (PrimaryURL, SDTMversion, "/datasets/", DOMAIN)
Response <- GET (endpoint, add_headers ("api-key" = myAPIkey, "accept" =
                                         "application/json"))
responseBody <- httr::content (response, as = "text", encoding = "UTF-8")
```

- **paste0()** function concatenates strings without gaps
- **httr** function "**content ()**" converts the response to a JSON format.



[Slide 7] JSON Response Body is a Single Large Text with "rootItem" Landmarks

```
Term", "type": "SDTM Dataset Variable"}, "rootItem": {"href": "/mdr/root/sdtmig/d  
AE.AEMODIFY", "type": "Root Data Element"}, "self": {"href": "/mdr/sdtmig/3-2/dat  
Variable"}}. "core": "Perm", "description": "If AETERM is modified to facilitate  
Term", "name": "AEMODIFY", "ordinal": "9", "role": "Synonym Qualifier", "simpleData  
LLT", "title": "Lowest Level Term", "type": "Class Variable"}, "parentDataset": {"  
Dataset"}, "parentProduct": {"href": "/mdr/sdtmig/3-2", "title": "Study Data Tabu  
(Final)", "type": "Implementation Guide"}, "priorVersion": {"href": "/mdr/sdtmig/  
Variable"}, "rootItem": {"href": "/mdr/root/sdtmig/datasets/AE/variables/AELLT"  
Element"}, "self": {"href": "/mdr/sdtmig/3-2/datasets/AE/variables/AELLT", "titl  
Variable"}}. "core": "Exp", "describedValueDomain": "MedDRA", "description": "Dict  
Term", "name": "AELLT", "ordinal": "10", "role": "Variable Qualifier", "simpleDatat  
LLTCD", "title": "Lowest Level Term Code", "type": "Class variable"}, "parentData  
Dataset"}, "parentProduct": {"href": "/mdr/sdtmig/3-2", "title": "Study Data Tabu  
(Final)", "type": "Implementation Guide"}, "priorVersion": {"href": "/mdr/sdtmig/  
Variable"}, "rootItem": {"href": "/mdr/root/sdtmig/datasets/AE/variables/AELLTC  
Element"}, "self": {"href": "/mdr/sdtmig/3-2/datasets/AE/variables/AELLTCD", "ti
```



[Slide 8] Split the JSON Response with the STRSPLIT() Function and the “rootItem” String Pattern as Separator

```
jsplits <- strsplit (paste0 (respBody), split='}', "rootItem":{'',  
fixed=TRUE, useBytes=FALSE)
```

- *convert back to vector with the UNLIST function:*

```
jsx <- unlist (jsplits)
```

- *obtain number of split segments (determined by number of variables)*

```
jsn <- length (jsx)
```

- *will use for iteration:*

```
maxrows <- jsn[[1]]
```



[Slide 9] For Each Variable, the Attributes DESCRIPTION and NAME are Targeted

- Even though each split segment contains variable attributes *description, name, ordinal, label, core, role, and simpleDatatype*, just NAME and DESCRIPTION will be extracted into a table, for the sake of simplicity
- 52 Variables are obtained from AE
- The first variable (STUDYID) is processed first into a single row table.
This is because the first segment has JSON header information in addition to the variable attributes and requires additional processing.
- The remaining 51 variables are then processed by iteration.



[Slide 10] What Are Regular Expressions?

- A sequence of characters that define a search pattern, mainly for use in pattern matching with strings and extraction.
- Meta characters have special meaning:

Expression	Description
*	Item occurs zero or more times
+	Item occurs one or more times
?	Item occurs zero or one time
{A}	Item occurs A number of times
{A,B}	Item occurs between A and B times
.	Any character
.*	Matches zero or more of any character

- POSIX Character Classes are written with square brackets and keywords surrounded by colons.

For example `[:print:]` would match space, alphabetic and punctuation characters.



[Slide 11] Use Regular Expression Functions *regexexec()* and *regmatches()* to Extract Variable Names and Descriptions – for 1st variable

- *Extract column names:*

```
jscolsx <-  
regexexec('[[[:print:]]+,\\\\"(description|describedValueDomain)\\\\":\\\\".+\\\\",[[:pri  
nt:]]+,\\\\"(name)\\\\":\\\\".+\\\\",[[:print:]]*',  
jsx[[2]], ignore.case = FALSE, perl = TRUE, useBytes = FALSE)
```

- *Extract values:*

```
jsrowsex <-  
regexexec('[[[:print:]]+,\\\\"description|describedValueDomain\\\\":\\\\"(.+)\\\\",[[:pri  
nt:]]+,\\\\"name\\\\":\\\\"(.+)\\\\",[[:print:]]*',  
jsx[[2]], ignore.case = FALSE, perl = TRUE, useBytes = FALSE)
```



[Slide 12] Assemble the Extracted info into a Table

- *regmatches()* uses the locations obtained by *regexec()* to extract text

```
colex <- regmatches(jsx[[2]], jscolsx)
```

```
rowex <- regmatches(jsx[[2]], jsrowsx)
```

- *matrix()* function converts the extracted values into a 1x2 matrix

- *data.frame()* converts the matrix to a dataset table

- *colnames()* assigns the extracted column headers to the columns of the data frame

```
jsmat <- matrix(data=rowex[[1]][2:3], nrow=1, ncol=2, byrow=TRUE)
```

```
jsmatrix <- data.frame(jsmat)
```

```
colnames(jsmatrix) <- unlist(colex[[1]][2:3])
```



[Slide 13] Repeat Process for Remaining Variables

- Rows 3 to maxrows are processed into a table called "jsframe"

```
for (n in 3:maxrows){  
  jsrows <-  
  regexec('[:print:]+,\\\"descri\\w{5,14}\\\".:\\\"(.+)\\\"',[:print:]+,\\\"name\\\".:\\\"(\\w{3,8})\\\"',[:pr  
int:]]+', jsx[[n]], ignore.case = FALSE, perl = TRUE, useBytes = FALSE)
```

```
  rowextrax <- regmatches(jsx[[n]],jsrows)  
  jsmax <- matrix(data=rowextrax[[1]][2:3], nrow=1, ncol=2, byrow=TRUE)  
  jsframe <- data.frame(jsmax)  
  colnames(jsframe) <- unlist(colex[[1]][2:3])
```

- the assembled table "jsframe" is appended to the previously created single-row table: "jsmatrix"

```
  jsmatrix <- rbind.data.frame(jsmatrix,jsframe)  
}
```



[Slide 14] Final DataFrame Table

description	name
Is this a serious event?	AESER
Internal or external identifier such as a serial number on an SAE reporting form.	AEREFID
If AETERM is modified to facilitate coding, then AEMODIFY will contain the modified text.	AEMODIFY
Identifies the end of the event as being before or after the reference time point defined by variable AEENTPT.	AEENRTPT
Identifier used to uniquely identify a subject across all studies for all applications or submissions involving the product.	USUBJID
Do additional categories for seriousness apply?	AESMIE
Did the serious event result in persistent or significant disability/incapacity?	AESDISAB
Did the serious event result in death?	AESDTH
Did the serious event require or prolong hospitalization?	AESHOSP
Did the serious event occur with an overdose?	AESOD





Thank You !

joehinson@outlook.com

