Paper SI02

Implementing the CDISC Library API in software applications: first experiences

Jozef Aerts, XML4Pharma, Tarrenz, Austria

ABSTRACT

The new released CDISC Library RESTful web services API opens many opportunities for automation of otherwise cumbersome tasks when using the CDISC standards. Copy-and-paste from IGs is no longer necessary, as is the use of Excel worksheets to keep track of standards versions.
The use of RESTful web services allows to obtain detailed information about submission domains, variables and controlled terminology. For example, software can do a look up whether a new version of an IG or controlled terminology was made available, and immediately install that and/or make it available without that the software itself need to be updated.
First experiences with implementing the CDISC Library RESTful web services API in different computer languages (Java, XSLT/XPath, XQuery and others) are presented.

INTRODUCTION

The CDISC Library API (formerly known as "SHARE API") is now available since April 2019 [1]. It allows to retrieve CDISC standards content using a set of RESTful web services as well as querying the library for specific pieces of information. As the consumption of RESTful web services is system and computer language neutral, any modern software can use the CDISC Library API and use the information in any system.
The CDISC Library functions as "the CDISC truth", so that it is essential that systems that implement the CDISC standards use the library as the "single source of truth". One can think here about systems like validators and mapping tools.

RESTFUL WEB SERVICES

RESTful web services are nowadays state of the art for retrieving information from large data repositories and for data exchange between systems over the internet (or intranet). They have the advantage that a single source of information can be used that is centrally maintained and updated, among others meaning that software must not be adapted or updated when the information that it uses is updated. Well know RESTful web services in healthcare are those of the National Library of Medicine, and of the National Cancer Institute. A good overview can be found in the article "The Use of RESTful web services in Medical Informatics and Clinical Research and Its Implementation in Europe" [2].
Also the new HL7-FHIR standard [3] for electronic health records is fully based on RESTful web services, allowing to combine information from different sources such as EHR systems from different hospitals.
RESTful web services can be used independently of the computer or operating system involved and can be implemented using any modern computer language such as Java, C#, C++, Python, XSLT, XQuery, etc..
Some RESTful web services require authentication (in healthcare especially when the information involves patient data), others are completely open for use by anyone without any authentication. This is important as the CDISC Library RESTful web services require "basic authentication" [4] which has some implications, as will be explained further on.

THE CDISC LIBRARY: BASIC PRINCIPLES OF USE

Currently, the CDISC Library only implements the "GET" method of the HTTP methods, meaning that information can only be retrieved, but not be "uploaded". This might change in future when it evolves into a "self-learning system".
The use of the CDISC library API requires an account that needs to be requested separately – it is not directly coupled to the membership account [1]. The user then obtains a username and a password. These can be used when testing the API in the browser, but will also be needed to be added to the HTTP header when using the API from within software. Username and password are personal, so should not be hardcoded in software applications that are distributed. In such a case, the end users will need their own username and password that are read in by the software, e.g. from a text file or by a user prompt.
The license agreement allows to distribute software (commercial or non-commercial) that uses the CDISC Library API, but in the case of commercial software requires some license fee payments to CDISC are due in the order of

less than about 10% of the yearly license fee that is charged to the end user. Also in that case, the end user must have its own account and password. In the case of free / open source software, no license fees apply.
The RESTful web services follow the HATEAOS principles [5]. Essentially, this means that the response from the CDISC Library server contains instructions ("links") that allow to use the obtained information in new queries. Implementation of HATEAOS is very important, as it allows systems to "drill down" or "walk up" into the "hierarchy of information". A simple example is that a query response that contains an SDTM domain name, also contains a reference for querying what SDTM variables can be used in that domain, and a reference for querying in which SDTM-IG versions that domain can be used.
The primary format the information is returned in by the RESTful web services is JSON. However, it is also possible to request the information to be returned as XML. A JSON-LD implementation for linked data may be added in the future. This is interesting, as the underlying database system is based on Graph technology.
At this moment, there are bandwidth limitations in the use of the CDISC Library API: 1.5-3 GB of traffic per month is possible depending on the membership status and commercial – non-commercial use [1]. Additional bandwidth can however be purchased at very low cost.

BASICS OF THE API

We will not go into the details of the API here as it is very well documented in the CDISC website [6]. Most important is the "base" of the services, which is:

```
library.cdisc.org/api
```

For security reasons (also as username and password are passed with the HTTP header), the "https" schema is used. So each request string will start with:

```
https://library.cdisc.org/api
```

and will be followed by methods and parameters. For example, to retrieve all information regarding the SDTM codelist "LBTESTCD" version 2019-03-29, the query string is:

```
https://library.cdisc.org/api/mdr/ct/packages/sdtmct-2019-03-29/codelists/C65047
```

where "C65047" is the NCI identifier of the codelist "LBTESTCD".

The nice thing about the RESTful web services concept is that such query strings can be tested in the browser. In the CDISC Library API case, the user will then be asked for his/her credentials, just as when logging in into a website.

SOFTWARE IMPLEMENTATIONS

In first instance, we were interested in implementations into three computer languages: Java, XSLT and XQuery. Java is the major computer language used for the development of software in our company, with software products that are either commercial or free and open source [7]. These include validation software, CDISC-SDTM mapping software, and software for designing study designs according to the CDISC standards, for generating define.xml files for electronic submissions to the regulatory authorities, and "smart visualization software" for as well collected data as for SDTM/SEND/ADaM submissions [8]. The reason we wanted to start implementing the CDISC Library API into such software packages is that we consider the CDISC Library as "the single source of truth" for CDISC standards. Furthermore, it would allow us to avoid to have to update our software each time a new version of CDISC standard is published. Examples are new versions of the SDTM-IG or SEND-IG or of CDISC controlled terminology.
Our interest in applying the CDISC Library API in XSLT is that our "SDTM-ETL" mapping software [9] uses XSLT to execute mappings between operational data and SDTM datasets. The XSLT is generated from an easy-to-learn propriety mapping language, for which a very large number of "wizards" have been implemented. So, for 90%, the user just uses graphical mapping wizards, but at the end, the transformation to SDTM (or SEND) us executed using XSLT. The CDISC Library API could then be used to help making decisions during the mapping process. For example, it can be used to find out whether an SDTM variable is "required", "expected" or "permissible", without the need of having a template available for the SDTM-IG version, or to find out under what conditions a timing or other

variable may be added to a domain. At the end, it should be possible to implement a new SDTM-IG or SEND-IG version automatically without any update necessary to the software itself.
Our interest in an XQuery implementation is that we (together with some other CDISC volunteers) started working on a completely open and transparent implementation of all validation rules published by the FDA, the PMDA and CDISC itself: the "Open Rules for CDISC Standards" initiative [10]. Goal of this initiative is to provide all the rules in a form that is completely open (i.e. the users can inspect how each rule is exactly implemented), that is independent of any software (users can develop their own software to execute the rules), and that can be executed by any software in any modern computer language. At this moment, XQuery is a candidate language for expressing such rules. In one of the pilots, each rule comes as an XQuery script that is versioned (i.e. each rule is versioned) [10], and RESTful web services can be used to retrieve or update the rules from a central server without the need of a software update. Those using the "classic" validation tools know how frustrating it is having to wait (sometimes for years) until a bug in a validation rule is fixed and implemented in a new version. With such an XQuery implementation of the "Open Rules", an incorrect implementation of a validation rule can be repaired within hours and made available through a RESTful web service immediately. But also the rules themselves can use RESTful web services to obtain information, e.g. to check the validity of a SNOMED-CT term, whether an SRS code and term (decode) really belong together, or to check the validity of a LOINC code, or whether the LOINC-to-SDTM mapping has been executed correctly. A good number of RESTful web services from different institutions (like the NLM) have already been implemented in our XQuery implementations, but an implementation of the CDISC Library API was still lacking. Especially for validation, the potential is enormous.

In each case, we have donated snippets of code to the "how to" pages on the "CDISC Wiki" [11]
Explaining how we implemented for each of the given languages. This wiki page also contains snippets for other languages such as Python and SAS.

JAVA IMPLEMENTATION

Java is a very popular computer language (but Python coming up enormously), among others due to the availability of a huge number of free or open source libraries.
For the use of RESTful web services, the Oracle/Eclipse Jersey libraries [12] are the most popular. For our first implementations, we used the Jersey-2 libraries which can be downloaded from GitHub [13].

Using Jersey-2, implementing simple "GET" RESTful web services such as the CDISC Library API only requires a few lines of code.
For our method to obtain the details of the "LBTESTCD" codelist version 2019-03-29:

```
import javax.ws.rs.client.*;  // Client and ClientBuilder - javax.ws.rs-api-2.1.jar
import javax.ws.rs.core.*;
…
// initialization and authentication
ClientConfig clientConfig = new ClientConfig();
client = ClientBuilder.newClient(clientConfig);
// Basic Authentication
HttpAuthenticationFeature feature =
HttpAuthenticationFeature.basic(username,pass);
client.register(feature);
…
// executing the query
String response;
WebTarget webTarget =
   client.target(base).path("mdr").path("ct").path("packages")
    .path(packageVersion).path("codelists").path(codeListNCICode);
Invocation.Builder invocationBuilder =
     webTarget.request(MediaType.APPLICATION_XML);
Response response = invocationBuilder.get();
System.out.println("HTTP Response Status = " + response.getStatus());
```

```
String xml = response.readEntity(String.class);
System.out.println(xml);
```

With (for our example), the variable "packageVersion" having the value "**sdtmct-2019-03-29**" and the variable "codeListNCICode" having the value "C65047". Of course, such queries should always be packed into reusable methods, which we have done for a large number of cases.
It is always a good idea to check the HTTP response code, which is "200" in the case the request was successful. All other response codes are explained on the CDISC Library website [14] and in many articles on the internet.

The first few lines demonstrate the initialization and the authentication mechanism ("Basic" authentication), with "username" and "pass" being the by CDISC provided username and password. Of course these should never be hardcoded into the source code, but e.g. be read from file or prompted for to the user.

In the example, XML is asked to be returned, which will need to be parsed, e.g. using a SAX parser [15].
When the line String "MediaType.APPLICATION_XML" is replaced by "MediaType.APPLICATION_JSON", then JSON formatted information will be returned, which is the default.

The first few lines of the response in case of an XML formatted response is:

```
<?xml version="1.0" encoding="utf-8"?>
<cdiscLibrary>
    <codelist>
        <conceptId>C65047</conceptId>
        <extensible>true</extensible>
        <name>Laboratory Test Code</name>
        <submissionValue>LBTESTCD</submissionValue>
        <definition>Terminology used for laboratory test codes of the CDISC Study Data Tabulation
            Model.</definition>
        <preferredTerm>CDISC SDTM Laboratory Test Code Terminology</preferredTerm>
        <synonyms>Laboratory Test Code</synonyms>
        <_links>
            <self>
                <href>/mdr/ct/packages/sdtmct-2019-03-29/codelists/C65047</href>
                <title>CDISC SDTM Laboratory Test Code Terminology</title>
                <type>Code List</type>
            </self>
            <parentPackage>
                <href>/mdr/ct/packages/sdtmct-2019-03-29</href>
                <title>SDTM Controlled Terminology Package 37 Effective 2019-03-29</title>
                <type>Terminology</type>
            </parentPackage>
            <rootItem>
                <href>/mdr/root/ct/sdtmct/codelists/C65047</href>
                <title>Version-agnostic anchor resource for codelist C65047</title>
                <type>Root Value Domain</type>
            </rootItem>
            <priorVersion>
                <href>/mdr/ct/packages/sdtmct-2018-12-21/codelists/C65047</href>
                <title>CDISC SDTM Laboratory Test Code Terminology</title>
                <type>Code List</type>
            </priorVersion>
        </_links>
```

Where it is interesting to see how the HATEAOS principle has been implemented in the "href" elements. For example, a "link" to the prior version of the codelist (version 2018-12-21) being provided.

Very recently, also a "search" functionality has been added to the API [16]. It allows to pass a search term and zero or more scopes for the search. Rather than a simple query string, it uses query parameters. For example, to search

about "LBTESTCD" in the scope of the "domain" "LB", and in the scope of the "product" "SDTMIG v3.3", the query string is:

https://library.cdisc.org/api/mdr/search?q=LBTESTCD&domain=LB&product=SDTMIG%20v3.3

where "%20" is the escape for the blank character.

In the Java code, this is established by using the "queryParam" method, once for the question itself ("q") and one or more times for the "scopes". For example:

```
webTarget = webTarget.queryParam("q", "LBTESTCD");
webTarget = webtarget.queryParam("domain", "LB");
webTarget = webTarget.queryparam("product", "SDTMIG v3.3");
```

A list of allowed "scopes" (like "domain", and "product") can be asked for using the method "/mdr/search/scopes".

At the moment of writing (August 2019), only JSON formatted information is returned. For example:


```
{ ⊟
   "hits":[ ⊟
      { ⊟
         "core":"Required",
         "product":"SDTMIG v3.3",
         "domain":"LB",
         "name":"LBTESTCD",
         "description":"Short name of the measurement, test, or examination described in LBTEST. It can be used as
         "datasetStructure":"One record per lab test per time point per visit per subject",
         "href":"/mdr/sdtmig/3-3/datasets/LB/variables/LBTESTCD",
         "label":"Lab Test or Examination Short Name.",
         "type":"SDTM Dataset Variable",
         "class":"Findings",
         "uiHref":"/mdr/ui/sdtmig/3-3/datasets/LB",
         "simpleDatatype":"Char"
      }
   ],
   "totalHits":1
}
```


More information about the different "search" methods can be obtained from the CDISC Library Wiki [16].

We expect that this "search" functionality will be very important in tools such as validator and mapping tools, as it is a very direct and compact way to obtain all primary information of e.g. SDTM variables.

XSLT IMPLEMENTATION

Our interest in implementing the CDISC Library API in XSLT lies in that we already use a lot of web services in our "SDTM-ETL" mapping software [9], where mapping scripts, usually generated by wizards, are translated into XSLT scripts that transform collected (e.g. from CRFs) operational data into SDTM or SEND datasets, first in CDISC Dataset-XML format, followed by a postprocessing "downgrading" step into SAS-XPT format.
RESTful web services can easily be implemented in XSLT, as the result of a query is nothing else than an XML document (the XSLT process will always request XML) which can be parsed in the XSLT itself. A simple example of such a query used in our SDTM-ETL software is:

<xsl:variable name="LOINComponent"
select="doc(http://www.xml4pharmaserver.com:8080/LOINCService/rest/LOINCCoreProperties/1751-7)//Component
returning the "component" for LOINC code "1751-7" and storing it in the XSLT variable "LOINCComponent" which can then be further used in the XSLT. Such a RESTful web service has also been developed by us for the famous "LOINC to CDISC mapping" (at the time of writing, in internal review at CDISC).
The CDISC Library API RESTful web services could also be implemented easily in XSLT if not for the authentication. CDISC uses the "Basic Authentication" mechanism, which is not directly supported by the XSLT standard. The EXPATH-XSLT extension libraries [17] do have a function "http-request" function that allows to pass username and password using basic authentication. Unfortunately, this EXPATH function is not supported by the major XSLT processors yet. As some XSLT processors are open source, such as Saxon-HE [18], it should be possible to extend these with the "http-request" functionality, but we haven't tried this yet.
An authentication mechanism that is sometimes used in RESTful web services is to pass username and password as request parameters, but according to CDISC, there are no plans to do so, as it is not very safe.

When using Saxon-HE as an XSLT processor, there is however a possibility to extend XSLT with own functions by adding Java code that implements them. The mechanism to do so is described at the Saxon website [19 - https://www.saxonica.com/html/documentation/extensibility/integratedfunctions/ext-simple-J.html]. For example, one can define an own XSLT function with 3 parameters:

cdisc:sdtmvariable(sdtm-ig_version, domain, variable_name)

with the XSLT prefix "cdisc" coupled to the namespace "http://cdisc.org/ns/cdisc-library" (one can use another namespace name as long as one remains consequent). In XSLT, the function can then be called e.g. by:

```
<xsl:variable name="variableproperties" select="cdisc:sdtmvariable('3-
2','PE','PESTRESC')" />
```

returning an XML structure from which information can be retrieved using XPath. One must then write a Java class that implements this function and implements the basic authentication and that then calls the CDISC Library function:

/mdr/sdtmig/{sdtm-ig_version}/datasets/{domain}/variables/{variable_name}

where the parts in curly brackets correspond to the 3 XSLT parameters.

This means that currently, there is no direct way to use the CDISC Library API within XSLT. One can however define one own XSLT functions (in a different namespace) and then connect this to the CDISC Library API through a Saxon extension Java class that takes care of the authentication.
We are currently developing a number of such extension functions for use in our "SDTM-ETL" mapping software, realizing that the CDISC Library is the "single source of truth".

XQUERY IMPLEMENTATION

What applies to XSLT also applies to XQuery: although XQuery supports the use of RESTful web services, there is no XQuery processor available yet that supports basic authentication for RESTful web services. We do already use a lot of RESTful web services in piloting the use of XQuery in the "Open Rules for CDISC Standards" initiative for real open and transparent validation rules. These currently use RESTful web services that are our own ones (but we would like to replace them by CDISC Library RESTful web services), and NLM UNII and SRS RESTful web services. Also here, it would be possible to define and implement extension functions, but this would violate our won set principle that the rules implementations must be independent of the software that uses the rules.
So, for an implementation of the CDISC Library API in the XQuery pilot implementation for the "Open Rules for CDISC" initiative, we will need to wait until the XPath/XQuery standard supports basic authentication for RESTful web services and libraries for this become available. There is some hope for this, as several large native XML database vendors have developed such a (propriety) implementation, and the W3C is considering to make this part of the standard [20]. Good news is however already that the latest XQuery version (3.1) supports JSON formatting and parsing, as more and more RESTful web services only return JSON-formatted responses.

OTHER IMPLEMENTATIONS

At the time of writing (September 2019), we know of only one software package that is generally available that uses the CDISC Library API RESTful web services, which is our own open source "Smart Submission Dataset Viewer" [8]. We expect however that many more will follow.
Regarding other computer languages, CDISC already published "how to" pages [11] with some sample code for SAS (v.9.4) and for Python. As these are both very popular languages in clinical research, we expect the number of implementations of the CDISC Library API to grow rapidly.

CONCLUSION

The CDISC Library API and its RESTful web services are a great way to gain access to the "CDISC truth" in software applications. We found out that implementation in Java programs is very easy, whereas the lack of support for "Basic HTTP authentication" in XSLT and in XQuery make the implementation more complicated for these languages. For XSLT, we are currently developing a set of Java "bridging" classes that allow to define our own extension XSLT functions that use the CDISC Library API, which will be implemented in our SDTM-ETL mapping software. For XQuery, this would also be possible, but for an XQuery implementation in the "Open Rules for CDISC Standards" initiative, we decided to not do so as we want the rules themselves to remain completely independent of any software.

REFERENCES

1. CDISC Library: https://www.cdisc.org/cdisc-library
2. J. Aerts, Stud Health Technol Inform. 2017; 236: 80-87.
3. HL7 FHIR: https://www.hl7.org/fhir/
4. RESTful API Authentication Basics: https://blog.restcase.com/restful-api-authentication-basics/
5. HATEOAS Driven REST APIs: https://restfulapi.net/hateoas/
6. CDISC Library API Documentation: https://www.cdisc.org/cdisc-library/api-documentation
7. The Java Programming Language: https://www.java.com/en/
8. The "Smart Submission Dataset Viewer": https://sourceforge.net/projects/smart-submission-dataset-viewe/
9. The SDTM-ETL™ mapping software: http://www.xml4pharma.com/SDTM-ETL/
10. The "Open Rules for CDISC Standards" Initiative: http://xml4pharmaserver.com/OpenRulesForCDISCStandards/index.html
11. Getting Started: Programmatically Connect to the CDISC Library API: https://wiki.cdisc.org/display/LIBSUPRT/Getting+Started%3A+Programmatically+connect+to+CDISC+Library+API
12. Project Jersey: https://en.wikipedia.org/wiki/Project_Jersey
13. Eclipse Jersey: https://eclipse-ee4j.github.io/jersey/
14. CDISC Library API Status Codes: https://wiki.cdisc.org/display/LIBSUPRT/HTTP+Status+Codes
15. Parsing an XML file using SAX: https://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html
16. CDISC Library API: Search: https://wiki.cdisc.org/display/LIBSUPRT/Search
17. EXSLT Module 2.0 – HTTP Client: http://expath.org/spec/http-client/20090112
18. Saxon-HE: http://saxon.sourceforge.net/
19. Saxon Java Extension Functions: https://www.saxonica.com/html/documentation/extensibility/integratedfunctions/ext-simple-J.html
20. HTTP Client Module 2.0: Draft Community Group Report 04 January 2019: http://expath.github.io/expath-cg/specs/http-client-2/

ACKNOWLEDGMENTS

Special thanks are due to Anthony Chow, Sam Hume and Sally Cassells of CDISC for their support during our initial attempts understanding and implementing the CDISC Library API.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Jozef Aerts - XML4Pharma
Email: Jozef.Aerts@XML4Pharma.com
Web: www.XML4Pharma.com – www.XML4PharmaServer.com