PharmaSUG 2019 - Paper PO-20

Tame Your SHARE with a PYTHON and SAS

Michael Stackhouse, Covance;

Terek Peterson, Covance;

ABSTRACT

Staying up to date with standards is of the utmost importance, and subtle changes can cause your data to deviate from compliance. With a little bit of SAS, and a little Python, you can easily automate the extraction of CDISC standards metadata using the new SHARE API. These standards files can then be used in SDTM and ADaM development to ensure that you have access to the latest metadata available for specification writing, quality control, and custom conformance checks. Embedded CDISC metadata opens doors of possibilities when available directly to your Programming team. This poster will explore how you can implement and automate this information to make sure your team never falls behind.

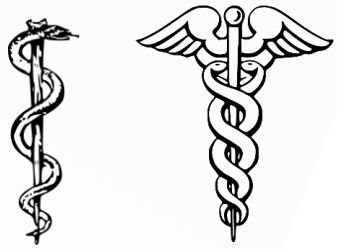
INTRODUCTION

A typical programming team within the Pharmaceutical industry has many SAS programmers and only a handful of programmers that know more contemporary programming languages and data set structures. Effectively programming only in SAS within a life altering new drug indication, for example, is applaudable. Manually importing standards into SAS can be tedious and labor intensive. So how can the use of a more contemporary programming language be leveraged to allow the large team of programmers' access to standards' metadata with in their known SAS environment?

CDISC has been providing standards' metadata for some time through SHARE. Though seemingly easy to access for some, SHARE is unattainable for most typical SAS programmers. CDISC has provided these standards' metadata by exporting views that are available at the Members Only area under CDISC SHARE Exports in Excel, ODM, RDF and PDF; however, not in any SAS data format. In the near future, CDISC will supply Biomedical Concepts, linking foundational standards with controlled terminology (CT), therapeutic area (TA) standards and other models. Access to SHARE will become integral to automation of standard mappings and other efficiencies.

THE CADUCEUS AND ROD OF ASCLEPIUS

Products derived from the bodies of snakes were known to have medicinal properties in ancient times. The Caduceus is a well-known medical symbol picturing a winged rod with two winding snakes. The Caduceus is used nowadays, especially in the USA, often as a symbol for healthcare organizations. Like the two snakes in this medical symbol, the two programming languages come together to create a solution for the important research we do every day that saves lives. The correct medical symbol of the Rod of Asclepius, with only a single snake, is associated with healing and medicinal arts. This reminds us to continue to research a solution using one programming language.



COVANCE
SOLUTIONS MADE REAL®

THE EXPERIMENT

Our basic hypothesis for this poster was with existing technologies available in our programming environment, the use of SHARE metadata can be extracted and provided as datasets to the existing programming staff in an automated process.

Use of the SHARE download site is a manual process that could be prone to errors when downloading. With the use of a Python script, this gives access to the SHARE API to pull metadata in XML format with the eye to the future in JSON format (more native to Python). JSON is a language-independent data format. Then a SAS batch program is executed to transform the downloaded metadata into SAS7BDAT format that can be accessed directly from an interactive SAS Enterprise Guide (EG) session. The process flow is shown below in Figure 1.

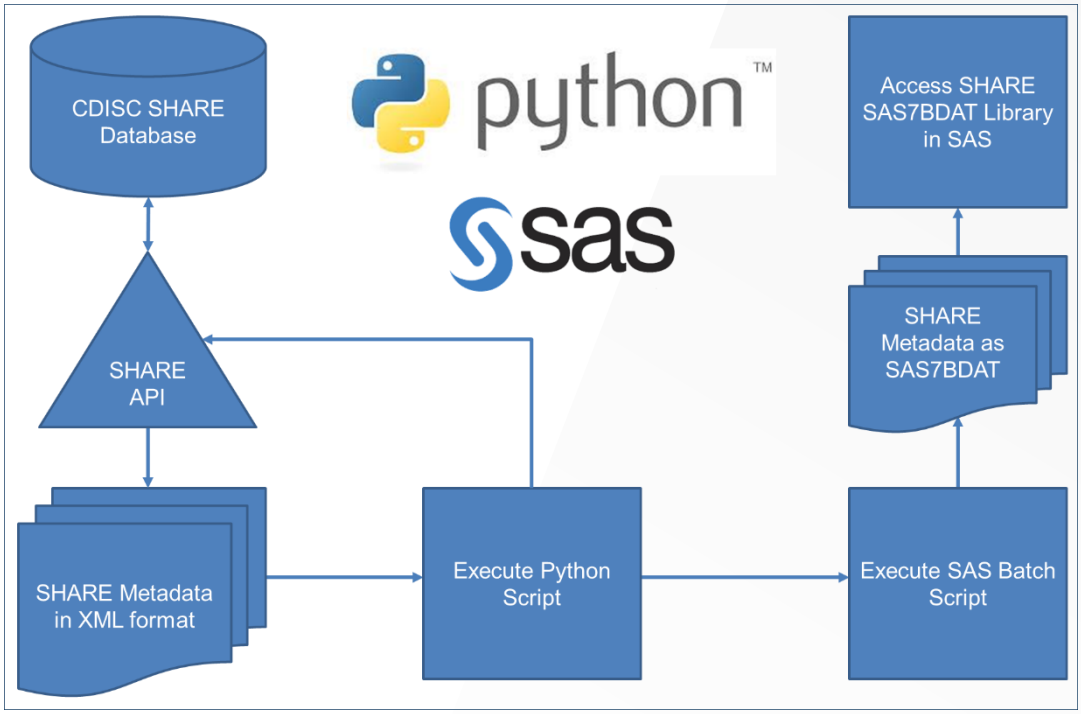


Figure 1. Python call to SHARE API to create SAS7BDAT Library

ADVANTAGES

Python offers several advantages while working with XML data. XML data does not need to be tabular in format, so Python is able to more specifically access and extract the information of interest. Furthermore, tools in Python allow you to parse an XML file without pulling the entire file into memory. With larger files,

COVANCE
SOLUTIONS MADE REAL®

this can dramatically speed up your programs and minimize the load on your server.

The advantages with SAS are that the largest number of programmers will be familiar with this programming environment. As can be seen in the SAS EG screen shot below, having CDISC standards' metadata, like SDTM AE, creates additional opportunities to the programmer. As CDISC provides additional metadata, this is just a "libref away" to that innovative program or macro. Additionally, this would not necessarily need IT system validation but could fall under existing programming processes easing implementation.

PYTHON CODE

```
import requests
import xml.etree.ElementTree as etree
import pandas as pd
import os

# Tag function
def fixtag(ns, tag, nsmap):
    ''' Fixes a specified namespace to an XML tag.'''
    return '{' + nsmap[ns] + '}' + tag

# Grab data from the URL
r =
requests.get("https://shareapi.cdisc.org/sm/rest/semantics/SHARE/domains/1.0_14055499822068020
44258/variables?user-id=bus_user&encoding=identity", auth=("<USERNAME>", "<PASSWORD>"))

# Write out the xml
with open('./tmp.xml', 'w+') as f:
    f.write(r.text)

# Iterate over the XML file and extract the desired data
for event, elem in etree.iterparse("./ae.xml", events=('start', 'end' , 'start-ns')):

    # Grab the namespaces from the XML file to make references easier
    if event == 'start-ns':
        ns, url = elem
        nsmap[ns] = url
        continue

    # Grab the domain information from the ItemGroupDef
    if event == 'start' and elem.tag == fixtag('', 'ItemGroupDef', nsmap):
        ItemGroupDef = elem.attrib
        ItemGroupDef['Domain Label'] = elem[0][0].text

    # Build the ItemRef table
    if event == 'start' and elem.tag == fixtag('', 'ItemRef', nsmap):
        try:
            ItemRef = ItemRef.append(pd.DataFrame(elem.attrib, index=[0]))
```

```
            except NameError:
                ItemRef = pd.DataFrame(elem.attrib, index=[0])
            continue

        # Build the detailed variable data into the ItemDef table
        if event == 'end' and elem.tag == fixtag('', 'ItemDef', nsmap):

            # Set null CDISC notes and NCI codes
            cdisc_notes, nci = '', ''

            # Grab nested variable information
            for child in elem:

                # Get the CDISC notes if available
                if child.tag == fixtag('mdr', 'CDISCNotes', nsmap):
                    cdisc_notes = child[0].text[1:] if child[0].text[0] == "\n" else child[0].text
                    # Strip leading message
                    cdisc_notes = ':'.join(cdisc_notes.split(":")[1:]).strip()

                # Get the NCI code from an Alias element
                if child.tag == fixtag('', 'Alias', nsmap):
                    if child.attrib['Context'] == 'nci:ExtCodeID':
                        nci = child.attrib['Name']

            tmp = pd.DataFrame(elem.attrib, index=[0])
            tmp['CDISC Notes'] = cdisc_notes
            tmp['NCI Code'] = nci

            # Compile into the ItemDef dataset
            try:
                ItemDef = ItemDef.append(tmp)
            except NameError:
                ItemDef = pd.DataFrame(tmp)

# Group all of the data into one table
domain = ItemRef.merge(ItemDef, how='outer', left_on='ItemOID', right_on='OID')

# Merge Domain level data onto table
for key, value in ItemGroupDef.items():
    domain[key] = value

# Remove namespace from column names
domain.columns = [x.replace('{' + nsmap['mdr'] + '}','') for x in domain.columns]

# Reorder columns on the table and keep only what's needed
domain = domain[['Domain','Domain Label', 'SASFieldName', 'DataType', 'SubmissionDataType',
'Role',
                 'Mandatory', 'OrderNumber', 'Repeating','NCI Code', 'CDISC Notes']]

# Convert OrderNumber to numeric
domain['OrderNumber'] = pd.to_numeric(domain['OrderNumber'])
```

```
# Sort by the OrderNumber
domain.sort_values('OrderNumber', inplace=True)

# Output
domain.to_csv(ItemGroupDef['Domain'] + '.csv', index=False)

# Clear the tmp XML
os.remove('./tmp.xml')
```

	Domain	Domain Lab...	SASFieldNa...	DataType	Submission...	Role	Mandatory	OrderNumb...	Repeating	NCI Code	CDISC Notes
1	AE	Adverse Events	STUDYID	text	Char	Identifier	Yes	1	No	C83082	Unique identifier for a study.
2	AE	Adverse Events	DOMAIN	text	Char	Identifier	Yes	2	No	C49556	Two-character abbreviation for the domain.
3	AE	Adverse Events	USUBJID	text	Char	Identifier	Yes	3	No	C70731	Identifier used to uniquely identify a subject across all studies for all applications or sub...
4	AE	Adverse Events	AESEQ	integer	Num	Identifier	Yes	4	No	C83213	Sequence Number given to ensure uniqueness of subject records within a domain. May...
5	AE	Adverse Events	AEGRPID	text	Char	Identifier	No	5	No	C83204	Used to tie together a block of related records in a single domain for a subject.
6	AE	Adverse Events	AEREFID	text	Char	Identifier	No	6	No		Internal or external identifier such as a serial number on an SAE reporting form.
7	AE	Adverse Events	AESPID	text	Char	Identifier	No	7	No		Sponsor-defined identifier. It may be pre-printed on the CRF as an explicit line identifier...
8	AE	Adverse Events	AETERM	text	Char	Topic	Yes	8	No	C78541	Verbatim name of the event.
9	AE	Adverse Events	AEMODIFY	text	Char	Synonym Qualifier	No	9	No	C83206	If AETERM is modified to facilitate coding, then AEMODIFY will contain the modified text.

Figure 2. SHARE Metadata in SAS

CONCLUSION

It is possible to use existing technologies readily available to a programming team to make CDISC SHARE metadata easily accessible through SAS. Python proved to be a good bridge because it offered several advantages while working with XML data. This exercise has shown it is possible to access the latest SHARE metadata to make available for specification writing, quality control, and custom conformance checks. Providing these metadata easily to SAS programming teams can bolster existing processes and create that next innovative program or macro. The authors know there are hundreds of ways to accomplish the same objective but found this way to be quick and easy.

REFERENCES

- https://www.cdisc.org/standards/share
- https://en.wikipedia.org/wiki/Rod_of_Asclepius
- https://en.wikipedia.org/wiki/Caduceus
- http://boscoh.com/programming/reading-xml-serially.html

CONTACT INFORMATION

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

Terek Peterson

Covance

1016 West Ninth Avenue

King of Prussia, PA / 19406

Email: Terek.Peterson@Covance.com

Michael Stackhouse

Covance

4000 Centregreen Way

Cary, NC / 27513

Email: Michael.Stackhouse@Covance.com

COVANCE
SOLUTIONS MADE REAL®