Paper 188-2015

Accessing and Extracting Unstructured XML Data using SAS and Python

Sai Mandagondi, Slalom Consulting;

ABSTRACT

This paper discusses an approach to dynamically load unstructured XML data using SAS and Python. When neither the SAS XML mapper nor a custom XML map can parse the incoming data, using external programs (Shell Scripting and Python) and integrating results from external programs into a SAS data set is an efficient alternative. One of the methods to eventually load data into a database to support upstream reporting and analytics is illustrated.

INTRODUCTION

Data exchange mediums like JSON, XML have garnered immense popularity over the last decade and play a central role in modern data communication techniques. SAS natively provides capabilities to make web service calls and download data from a given URL. XML data can be downloaded in SAS by either building a custom XML map or utilizing SAS XML mapper to define an XML structure. Goal of this paper is to propose a new method to access and extract XML data via SAS Enterprise Guide and load it into a Teradata database. Leveraging external tools and there by extending inherent capabilities of SAS are discussed in the paper using a sample XML which does not have an expected hierarchical structure. Also usage of native SAS PROC IMPORT and sample Python script to analyze the XML structure are demonstrated.

ELEMENTS OF DATA EXTRACTION

Fundamental to the whole process is to import data into a format and location that SAS can access, import and manipulate. Figure 1. elucidates the different steps involved in the data retrieval.

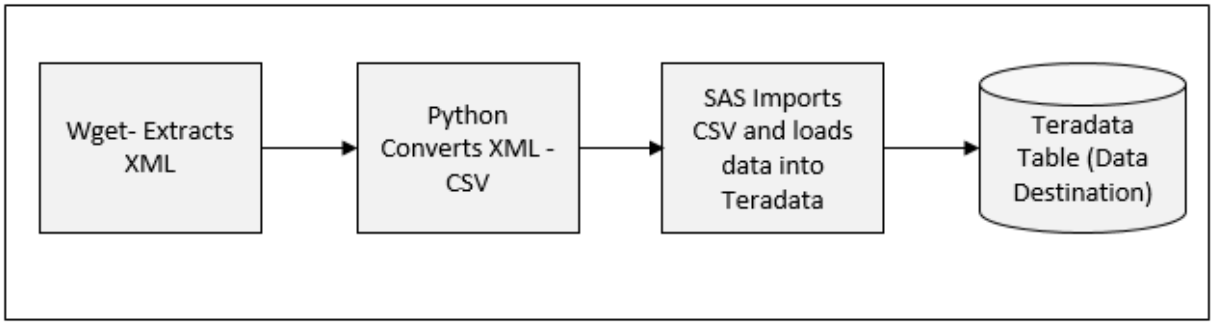

Figure 1Process Flow

Wget

Wget command helps download data from HTTP, HTTPS and FTP protocols. In shell scripting alongside Curl it is widely used to retrieve data from a web service call.

XML

XML (eXtensible Markup Language) is a widely used data transfer medium which is generally hierarchical in nature. Modern API requests and web service calls leverage XML based communication to transfer data.

SAS from version 8.2 and higher has provisions to handle XML data. Additionally SAS 9.1 and higher provide an XML Mapper to build XML maps. Challenge is to deal with unstructured or undefined XML data where markup tags are not hierarchical and convert it into SAS datasets.

Python – Packages and Libraries

Python is a multi-paradigm programming language (supports Object oriented and structured programming). For internet facing applications large number of standard formats and protocols (HTTP, MIME) are supported. One of the powerful features of Python is its ability to break non coherent data structures into manageable and easily consumable formats.

SAS PROC IMPORT

SAS PROC IMPORT can import wide variety of data sources including tab delimited, Comma delimited and excel files. It is a starting point for reading a delimited ASCII data file such as a CSV (comma-separated values). Premise of this alternate approach is to leverage PROC IMPORT and extract data from the CSV generated by Python. Adopted the following PROC IMPORT options when retrieving data from CSV in SAS Enterprise Guide. Multiple options can be set in PROC IMPORT procedure. Guessing Rows, Data Row are some of the most frequently used options.

```
/*Sample SAS PROC IMPORT procedure*/
PROC IMPORT
DATAFILE = FILEPATH;
DBMS = CSV /* In chosen example*/; REPLACE OUT = XYZ;
GUESSINGROWS = 32767;
```

DECONSTRUCTING, ACCESSING AND LOADING XML DATA

Shell Scripting - Downloading XML

Wget command fetches OrderShipping.xml file from a web service call and this XML is used as an input to a python program to convert into a csv. Generated output is then transported to a location that SAS can access and import. Leveraged Wget command as shown in Figure 2. to extract the OrderShipping.xml file from a URL by executing a shell script. Alternately same operation can be performed in SAS by running the above shell script from SAS Enterprise Guide. Rationale behind choosing to run a shell script has multiple advantages over directly making a web service call from SAS. It serves the dual purpose of making a web service call, downloading an XML file and running the python script.

```
#!/bin/bash
rm -f OrderShipping.xml
wget -P /dev/devmtc/manual_dims/ http://slalomsai.com/OrderShipping.xml -O OrderShipping.xml
python OrderShipping.py -x OrderShipping.xml -c OrderShipping.csv
```

Figure 2 Download XML file from a URL

Analyzing the XML

Consider the sample XML displayed in Figure3. It consists of iPhone 6 shipment information and shipment cycle (order of shipping) associated with each Shipment Number across different states. Similar Shipment

tags exist for different states. As the XML tags are not hierarchical in nature they cannot be imported directly into SAS by building a custom map or leveraging the XML mapper. Sample python code discussed in the paper converts the XML file into a comma-separated delimited file, popularly referred to as a CSV (comma-separated values). Perl or R can be used as substitutes for python to download data but the key point is to ensure that data is loaded into a location that SAS can access.

```
<?xml version="1.0" ?>
- <OrderShipments>
- <Shipment ProductName="iPhone 6" State="GA">
  <ShipmentLeg
             ShippingCycle="1234"
             ProductName="iPhone 6"
             ShipmentNumber="1706"
             OriginShippedDateTime="2015-07-21T12:28:00"
             ExpectedArrivalDateTime="2015-07-24T17:39:00"/>
  <ShipmentLeg ShippingCycle="3732"
               ProductName="iPhone 6"
               ShipmentNumber="333"
               OriginShippedDateTime="2015-07-21T18:34:00"
               ExpectedArrivalDateTime="2015-07-28T22:34:00"/>
  <ShipmentLeg ShippingCycle="3733"
               ProductName="iPhone 6"
               ShipmentNumber="1644"
               OriginShippedDateTime="2015-07-22T00:15:00"
               ExpectedArrivalDateTime="2015-07-25T05:20:00"/>
  <ShipmentLeg ShippingCycle="3734"
               ProductName="iPhone 6"
               ShipmentNumber="1406"
               OriginShippedDateTime="2015-07-23T06:38:00"
               ExpectedArrivalDateTime="2015-07-24T10:41:00"/>
  <ShipmentLeg ShippingCycle="3735"
               ProductName="iPhone 6"
               ShipmentNumber="2423"
               OriginShippedDateTime="2015-07-25T12:47:00"
               ExpectedArrivalDateTime="2015-07-26T17:29:00"/>
  <ShipmentLeg ShippingCycle="3736"
               ProductName="iPhone 6"
               ShipmentNumber="588"
               OriginShippedDateTime="2015-07-22T18:36:00"
               ExpectedArrivalDateTime="2015-07-29T21:47:00"/>
  <ShipmentLeg ShippingCycle="3737"
               ProductName="iPhone 6"
               ShipmentNumber="1505"
               OriginShippedDateTime="2015-07-23T22:55:00"
               ExpectedArrivalDateTime="2015-07-31T02:50:00"/>
  <ShipmentLeg ShippingCycle="3738"
               ProductName="iPhone 6"
               ShipmentNumber="2593"
               OriginShippedDateTime="2015-07-23T05:20:00"
               ExpectedArrivalDateTime="2015-07-26T09:59:00"/>
  </Shipment>
```

Figure 3. Sample Order Shipping XML

A closer examination of XML data indicates that ShipmentLeg is the repetitive element of the XML and each ShipmentLeg has child attributes like OriginShippedDate, ShipmentNumber etc. If State becomes the Parent Element of the XML tag we can decompose the XML into a hierarchical structure. Thus an unstructured XML file can be converted into a hierarchical structured that can be iterated for each state and ShipmentLeg combination.

Python to CSV conversion

Python deconstructs the XML file and converts it into a Comma-separated list of values which can be imported into SAS. As shown in Figure 4, python synthesizes the XML into a CSV. Initial step is to identify the parent element, child element and child element attributes.

Based on the XML breakdown and analysis in the decomposition phase, analyzed and concluded that there will be multiple shipment legs for each state. Utilized native CSV modules in Python to read the XML, iterate over each ShipmentLeg for each state and write data to a CSV file. Included the important elements of the python code to illustrate the breakdown of the unstructured XML.

```
#!/usr/bin/python

import sys, getopt

parentElementName = 'Shipment'
parentElementAttributeNames = ['State']
childElementName = 'ShipmentLeg'
childElementAttributeNames = [
    'ProductName',
    'ShipmentCycle',
    'ShipmentNumber',
    'OriginShippedDateTime',
    'ExpectedArrivalDateTime',
    ];

def main(argv):
    xmlFilePath = ''
    csvFilePath = ''
    try:
        opts, args = getopt.getopt(argv,"hx:c:",["xfile=","cfile="])
    except getopt.GetoptError:
        print 'OrderShipping.py -x <xmlfile> -c <csvfile>'
        sys.exit()
    for opt, arg in opts:
        if opt == '-h':
            print 'OrderShipping.py -x <xmlfile> -c <csvfile>'
            sys.exit()
        elif opt in ("-x", "--xfile"):
            xmlFilePath = arg
        elif opt in ("-c", "--cfile"):
            csvFilePath = arg
    if (xmlFilePath == '' or csvFilePath == ''):
        print 'OrderShipping.py -x <xmlfile> -c <csvfile>'
```

Figure 4. Python Code to Convert XML to CSV

Generating and Accessing CSV File

Python code above successfully builds a CSV file and places the output file in a path that SAS can access. This is another key element of the whole data transfer operation which promotes accurate and effective data import.

The CSV file generated is displayed in Figure 5. Observe that the XML structure at this point is totally converted into a CSV format with the child attribute elements becoming the file headers. Notice that for each State and ShipmentLeg combination Python made an entry to the CSV file.

	A	B	C	D	E	F
1	State	ProductName	ShipmentCycle	ShipmentNumber	OriginShippedDateT	ExpectedArrivalDateTi
2	GA	iPhone6	1234	1706	2015-07-21T12:28:00	2015-07-24T17:39:00
3	GA	iPhone6	3732	333	2015-07-21T18:34:00	2015-07-28T22:34:00
4	GA	iPhone6	3733	1644	2015-07-22T00:15:00	2015-07-25T05:20:00
5	GA	iPhone6	3734	1406	2015-07-23T06:38:00	2015-07-24T10:41:00
6	GA	iPhone6	3735	2423	2015-07-25T12:47:00	2015-07-26T17:29:00
7	GA	iPhone6	3736	588	2015-07-22T18:36:00	2015-07-29T21:47:00
8	GA	iPhone6	3737	1505	2015-07-23T22:55:00	2015-07-31T02:50:00
9	GA	iPhone6	3738	2593	2015-07-23T05:20:00	2015-07-26T09:59:00
10	CA	iPhone6	2724	3706	2015-07-21T12:28:00	2015-07-24T17:39:00
11	CA	iPhone6	2726	9813	2015-07-21T18:34:00	2015-07-28T22:34:00
12	CA	iPhone6	3145	1344	2015-07-22T00:15:00	2015-07-25T05:20:00
13	CA	iPhone6	3145	1496	2015-07-23T06:38:00	2015-07-24T10:41:00
14	CA	iPhone6	3247	2413	2015-07-25T12:47:00	2015-07-26T17:29:00
15	CA	iPhone6	5758	5188	2015-07-22T18:36:00	2015-07-29T21:47:00
16	CA	iPhone6	4747	15305	2015-07-23T22:55:00	2015-07-31T02:50:00
17	CA	iPhone6	1357	25913	2015-07-23T05:20:00	2015-07-26T09:59:00
18	TX	iPhone6	4724	706	2015-07-21T12:28:00	2015-07-24T17:39:00
19	TX	iPhone6	2826	4813	2015-07-21T18:34:00	2015-07-28T22:34:00
20	TX	iPhone6	3445	4344	2015-07-22T00:15:00	2015-07-25T05:20:00
21	TX	iPhone6	3245	9496	2015-07-23T06:38:00	2015-07-24T10:41:00
22	TX	iPhone6	3347	6413	2015-07-25T12:47:00	2015-07-26T17:29:00
23	TX	iPhone6	9758	9188	2015-07-22T18:36:00	2015-07-29T21:47:00
24	TX	iPhone6	4047	2305	2015-07-23T22:55:00	2015-07-31T02:50:00

OrderShipping

Figure 5 Python Output - OrderShipping.csv

SAS PROC IMPORT – Database Load

CSV file generated above is imported into SAS using PROC IMPORT procedure as a SAS dataset. The imported SAS data set is then loaded into a Teradata database via SAS. A predefined table structure can

be used as a placeholder to store incoming OrderShipping details. Alternately tables can be dynamically created and dropped in SAS as illustrated in Figure 6. Used the former to load data into a Teradata environment after importing the CSV into a SAS data set.

```
%include '/Dev/Devtmc/Code/login.sas';

*import the manual file maintained by cabin;
PROC IMPORT  datafile='/Dev/Devtmc/manualfiles/OrderShipping.csv'
     DBMS=csv replace OUT =OrderShipping;
     SHEET ='OrderShipping';
     GUESSINGROWS=10000;
RUN;
PROC SQL;
      DROP TABLE CM_TDSBX.Import_OrderShipping;
      CREATE TABLE CM_TDSBX.Import_OrderShipping AS
      SELECT
            State
           ,ProductName
           ,ShipmentCycle
           ,ShipmentNumber
           ,OriginShippedDateTime
           ,ExpectedArrivalDateTime
           ,(DATETIME()) FORMAT=DATEAMPM19. LABEL='ExtractDateTime' AS ExtractDateTime
      FROM OrderShipping
;

QUIT;
```

Figure 6 SAS PROC IMPORT

IMPORT_ORDERSHIPPING

Filter and Sort | Query Builder | Data | Describe | Graph | Analyze | Export | Send To

	State	ProductName	ShipmentCycle	ShipmentNumber	OriginShippedDateTime	ExpectedArrivalDateTime	ExtractDateTime
1	CA	iPhone6	2724	3706	21JUL2015:12:28:00	24JUL2015:17:39:00	23JUL2015:00:03:11
2	GA	iPhone6	1234	1706	21JUL2015:12:28:00	24JUL2015:17:39:00	23JUL2015:00:03:11
3	TX	iPhone6	4724	706	21JUL2015:12:28:00	24JUL2015:17:39:00	23JUL2015:00:03:11
4	CA	iPhone6	2726	9813	21JUL2015:18:34:00	28JUL2015:22:34:00	23JUL2015:00:03:11
5	GA	iPhone6	3732	333	21JUL2015:18:34:00	28JUL2015:22:34:00	23JUL2015:00:03:11
6	TX	iPhone6	2826	4813	21JUL2015:18:34:00	28JUL2015:22:34:00	23JUL2015:00:03:11
7	CA	iPhone6	3145	1344	22JUL2015:00:15:00	25JUL2015:05:20:00	23JUL2015:00:03:11
8	GA	iPhone6	3733	1644	22JUL2015:00:15:00	25JUL2015:05:20:00	23JUL2015:00:03:11
9	TX	iPhone6	3445	4344	22JUL2015:00:15:00	25JUL2015:05:20:00	23JUL2015:00:03:11
10	CA	iPhone6	3145	1496	23JUL2015:06:38:00	24JUL2015:10:41:00	23JUL2015:00:03:11
11	GA	iPhone6	3734	1406	23JUL2015:06:38:00	24JUL2015:10:41:00	23JUL2015:00:03:11
12	TX	iPhone6	3245	9496	23JUL2015:06:38:00	24JUL2015:10:41:00	23JUL2015:00:03:11
13	CA	iPhone6	3247	2413	25JUL2015:12:47:00	26JUL2015:17:29:00	23JUL2015:00:03:11
14	GA	iPhone6	3735	2423	25JUL2015:12:47:00	26JUL2015:17:29:00	23JUL2015:00:03:11
15	TX	iPhone6	3347	6413	25JUL2015:12:47:00	26JUL2015:17:29:00	23JUL2015:00:03:11
16	CA	iPhone6	5758	5188	22JUL2015:18:36:00	29JUL2015:21:47:00	23JUL2015:00:03:11
17	GA	iPhone6	3736	588	22JUL2015:18:36:00	29JUL2015:21:47:00	23JUL2015:00:03:11
18	TX	iPhone6	9758	9188	22JUL2015:18:36:00	29JUL2015:21:47:00	23JUL2015:00:03:11
19	CA	iPhone6	4747	15305	23JUL2015:22:55:00	31JUL2015:02:50:00	23JUL2015:00:03:11
20	GA	iPhone6	3737	1505	23JUL2015:22:55:00	31JUL2015:02:50:00	23JUL2015:00:03:11
21	TX	iPhone6	4047	2305	23JUL2015:22:55:00	31JUL2015:02:50:00	23JUL2015:00:03:11
22	CA	iPhone6	1357	25913	23JUL2015:05:20:00	26JUL2015:09:59:00	23JUL2015:00:03:11
23	GA	iPhone6	3738	2593	23JUL2015:05:20:00	26JUL2015:09:59:00	23JUL2015:00:03:11
24	TX	iPhone6	1957	8113	23JUL2015:05:20:00	26JUL2015:09:59:00	23JUL2015:00:03:11

Figure 7 SAS PROC IMPORT Output

As shown in Figure 6 and 7, SAS converts the CSV into a SAS data set. Output from the SAS data set is then loaded into the Teradata database environment. Output from Teradata SQL Assistant is shown in Figure 8.This data in turn supports upstream order tracking and shipping activities.

Answerset 1

	State	Product Name	ShipmentCycle	Shipment Number	OriginShippedDate Time	ExpectedArrivalDateTime	ExtractDateTime
1	CA	iPhone6	3,145.00	1,344.00	7/22/2015 00:15:00	7/25/2015 05:20:00	7/23/2015 00:03:11
2	CA	iPhone6	3,247.00	2,413.00	7/25/2015 12:47:00	7/26/2015 17:29:00	7/23/2015 00:03:11
3	CA	iPhone6	5,758.00	5,188.00	7/22/2015 18:36:00	7/29/2015 21:47:00	7/23/2015 00:03:11
4	CA	iPhone6	4,747.00	15,305.00	7/23/2015 22:55:00	7/31/2015 02:50:00	7/23/2015 00:03:11
5	CA	iPhone6	1,357.00	25,913.00	7/23/2015 05:20:00	7/26/2015 09:59:00	7/23/2015 00:03:11
6	CA	iPhone6	3,145.00	1,496.00	7/23/2015 06:38:00	7/24/2015 10:41:00	7/23/2015 00:03:11
7	CA	iPhone6	2,726.00	9,813.00	7/21/2015 18:34:00	7/28/2015 22:34:00	7/23/2015 00:03:11
8	CA	iPhone6	2,724.00	3,706.00	7/21/2015 12:28:00	7/24/2015 17:39:00	7/23/2015 00:03:11
9	GA	iPhone6	3,733.00	1,644.00	7/22/2015 00:15:00	7/25/2015 05:20:00	7/23/2015 00:03:11
10	GA	iPhone6	3,735.00	2,423.00	7/25/2015 12:47:00	7/26/2015 17:29:00	7/23/2015 00:03:11
11	GA	iPhone6	3,736.00	588.00	7/22/2015 18:36:00	7/29/2015 21:47:00	7/23/2015 00:03:11

Figure 8 Database Load Output

CONCLUSION

This paper shows you can use SAS and Python combination to for extracting and loading XML data. Integrating external programs (Shell Scripts (Wget Command), Python) with SAS can help you get the work done. Using external programs gives more flexibility to accommodate changes in underlying data or updates in data transfer mechanisms. The idea is to provide more avenues and multitude of options to access data and build dynamic data sets to extend the various capabilities of SAS.

REFERENCES

- Miriam Cisternas and Ricardo Cisternas, 2003, “Reading and Writing XML files form SAS®”, (Paper 119-29), Proceedings of the Twenty-Ninth Annual SAS Users Group International Conference, CARY, NC: SAS Institute Inc. http://www2.sas.com/proceedings/sugi29/119-29.pdf
- George Zhu, Sunita Ghosh, Alberta Health Services – Cancer Care,2012, “Accessing and Extracting Data from the Internet Using SAS®”, Paper 121-2012,Proceedings of SAS Global Forum, 2012, Orlando, FL http://support.sas.com/resources/papers/proceedings12/121-2012.pdf

CONTACT INFORMATION

Your suggestions, comments and questions are encouraged and valued.

Sai Mandagondi (saim@slalom.com)

ACKNOWLEDGEMENTS