Paper 075-2012

A Simple Way of Importing from a REST Web Service into SAS® in Three Lines of Code

Philip Busby, SAS Institute, Cary, NC, USA

ABSTRACT

Think old programmers can't learn new tricks? In this paper, I show two neat tricks that combine into something really clever. (1) Our first neat trick is that instead of a path to a file on disk, the FILENAME statement can also accept a URL (via the URL access method). (2) Second, the SAS XML LIBNAME Engine (SXLE) can read in a static XML document from a FILENAME statement and turn it into a table. Combining these two, you can have SAS download a dynamically generated XML document, probably produced by a REST (representational state transfer) Web service (which can be built in Java, PHP, or .NET; it doesn't matter), and you can do it all in three lines of code.

INTRODUCTION

One of the amazing things about SAS is its ability to bring in data from almost any source as data sets for processing and analytics. There seems to be a SAS/ACCESS offering for every database under the sun, and even a few more for databases that still live in caves. SAS has long been able to process text data, import CSV files, and thoroughly understand data from all directions. But, an external vendor probably is not going to give us access to their database.

More recently, Web services have become the access method of choice, and all of these APIs are most often either SOAP (Simple Object Access Protocol) or REST Web services. Web services offer a secure avenue of data into and out of sites like Twitter and Facebook. Microsoft Excel and Google spreadsheets have very easy function calls to get XML data. But, until now, methods of getting Web service data into SAS have been cumbersome and difficult. So I suggest a very simple way of doing that by combining the SAS XML LIBNAME Engine with a FILENAME statment URL access method.

DEFINITIONS, ACRONYMS, AND JARGON

Web Service

When SOAP rose into popularity, it became synonymous with Web service. By definition a Web service is just a program sitting on a network that is designed to be used by another program on the other end of the network. Messages do not have to be wrapped in an XML packet, nor do they even have to be XML packets. By definition, your e-mail server is a Web service.

SOAP

A protocol that enforces an XML grammar to standardize the format of Web services, SOAP is used to increase interoperability between businesses and organizations. It stands for Simple Object Access Protocol, but it can be more intimidating than it is simple.

REST

A looser protocol than SOAP, "RESTful" Web services piggy back on HTTP methods. While SOAP requests sent over HTTP are all POST requests, REST uses GET, PUT, POST, and DELETE to achieve "CRUD"-like operations (Richardson, 2007). REST benefits from HTTP-caching mechanisms at the network level as GET, PUT, and DELETE are all idempotent; meaning the same operation can be done multiple times with no side-effects. REST Web services are often used with AJAX Web applications.

AJAX

Asynchronous Javascript and XML(AJAX) is a technique that uses Javascript in to download XML data into a webpage after the page has loaded. It is similar to the old dynamic HTML technique. However, the XML that is downloaded is often dynamic output from a Web service.

CRUD

Create, Read, Update, Delete (CRUD) is the set of four common operations that almost all applications implement along with business logic.

XML

Extensible Markup Language (XML) is a markup language for hierarchical, cross-platform transmission of data that is

both human and machine-readable. It also includes the character set encoding, so diacritics as well as eastern-European and Asian scripts are mangled less often.

XSLT

A programming language, Extensible Stylesheet Language Transformation (XSLT), can tell an XSLT processor how to transform an XML document into different file formats, such as HTML, an Image, a PDF, a CSV, or even another XML document.

CSV

Comma-separated-value (CSV) is a file format commonly used to transport tables across systems. Tables must be flat.It lacks any encoding markers, so it is not as expressive as an XML document would be. However, creation and processing of them is much easier.

SXLE

SAS XML LIBNAME Engine (SXLE)processes an XML document using an XSL transformation to present read-only data sets to SAS. It is pronounced "6-L" (Friebel, 2005).

WEB SERVICES OVER HTTP

Every request that goes across the Internet is an HTTP request, where a client (for example your browser) makes a request to a server to GET a resource, and the server responds with HTTP 200 OK, and sends the file. Sometimes the client says, "Hey, can you give me that file if it has changed since yesterday?", and the server says, HTTP 304 Not Modified, "Oh, it is still the same." Sometimes the server says, HTTP 404 Not Found, "Oops, that file is not here." Sometimes your client posts the contents of a form. In that case, the client's request can have additional data following it. The server will then execute a script or program, and come back with an HTTP 200 OK, or maybe that program misbehaves. In that case, the client gets HTTP 500 Internal Server Error. What's important here is that requests on HTTP are generally the client saying "do this" or "give me this" and the server doing it. (Helf, 2005)

WHAT SOAP REQUESTS LOOK LIKE

When a program behaves as a browser, it can send SOAP packets embedded within HTTP POST requests. They are messages entirely wrapped within XML, which is entirely wrapped inside of HTTP's messages. For example, a request to a SOAP Web service to calculate Elo rating changes from the results of a chess match might look like this:

```
POST /soap/EloCalculator.php HTTP/1.1
Host: philihp.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
       soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://philihp.com/soap">
    <m:CalculateRequest>
      <m:playerAOldRating>1453</m:playerAOldRating>
      <m:playerBOldRating>1529</m:playerBOldRating>
      <m:kValue>32</m:kValue>
      <m:winner>B</m:winner>
    </m:CalculateRequest>
  </soap:Body>
</soap:Envelope>
```

And the response might look like this:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
               soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.example.org/stock">
```

```
    <m:CalculateResponse>
      <m:playerANewRating>1440</m:playerANewRating>
      <m:playerBNewRating>1542</m:playerBNewRating>
    </m:CalculateResponse>
  </soap:Body>
</soap:Envelope>
```

WHAT REST REQUESTS LOOK LIKE

REST services, on the other hand, rather than rely on fitting everything into a well-ordered XML schema, override HTTP methods. While 99.999% of HTTP requests on the Internet are either GET or POST requests (Hyperbole), HTTP has some other methods, such as PUT and DELETE. These relate somewhat to CRUD actions (Richardson, 2007). This is a REST request:

```
GET /eloRating.xml?playerAOldRating=1453&playerBOldRating=1529&kValue=32&winner=B HTTP/1.1
Host: philihp.com
```

The request could simply return this:

```
<?xml version="1.0">
<CalculateResponse>
  <playerANewRating>1440</playerANewRating>
  <playerBNewRating>1542</playerBNewRating>
</CalculateResponse>
```

The advantage here is that this request looks almost exactly like any request to download from a Web site, and SAS already has a facility to download from a Web site.

FILENAME STATEMENTS

The FILENAME statement is how SAS connects to a file system, or other operating system resources. When you use the FILENAME statement, the URL access methods, SAS downloads a file for you at run-time.

If you downloaded your tweets from Twitter to a file on your hard drive, you could read it into SAS and print it to the log using this code:

```
filename tweets url 'c:\tweets.xml';
data _null_;
   infile tweets length=len;
   input record $varying200. len;
   put record $varying200. len;
run;
```

Or, you could have SAS download a live stream of your tweets by doing this:

```
filename tweets url
'https://api.twitter.com/1/statuses/user_timeline.xml?screen_name=philihp';
data _null_;
   infile tweets length=len;
   input record $varying200. len;
   put record $varying200. len;
run;
```

As a side note, you can change the ".xml" in that request to be ".json", and you will get the same data back in a structure that Javascript can more easily process. But, since SAS is already so good with XML, we will stick with ".xml".

This can be over HTTP or HTTPS (SSL), as SAS automatically detect this. By doing this, you can be sure the transmission across the Internet is secure, and no one snooped your traffic.

SAS XML LIBNAME ENGINE

The SAS XML LIBNAME engine gives SAS users an easy way of reading XML files from their hard drive, transforming them using an XSLT-like XPATHy language, and reading them from SAS as if they were native SAS data sets. (Friebel, 2002)

If we wanted to process our downloaded tweets from before, we could do this:

```
filename sxlemap 'c:\tweets.map';
filename tweetlib 'c:\tweets.xml';
libname tweetlib xml xmlmap=sxlemap access=readonly;
data _null_;
  set tweetlib.tweets;
  put _all_;
run;
```

A file at c:\tweets.map looks like this:

```
<?xml version="1.0" encoding="windows-1252"?>
<SXLEMAP version="1.2">
    <TABLE name="tweets">
        <TABLE-PATH syntax="XPath">/statuses/status</TABLE-PATH>
        <COLUMN name="id">
            <PATH syntax="XPath">/statuses/status/id</PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>INT</DATATYPE>
            <LENGTH>8</LENGTH>
        </COLUMN>
        <COLUMN name="text">
            <PATH syntax="XPath">/statuses/status/text</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>140</LENGTH>
        </COLUMN>
        <COLUMN name="retweeted">
            <PATH syntax="XPath">/statuses/status/retweet_count</PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>INT</DATATYPE>
            <LENGTH>8</LENGTH>
        </COLUMN>
        <COLUMN name="created">
            <PATH syntax="XPath">/statuses/status/created_at</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>30</LENGTH>
        </COLUMN>
    </TABLE>
</SXLEMAP>
```

We would see a table at tweetlib.tweets with a table of our tweets. If the markup for this file feels cumbersome to at first, try using the SAS XML Mapper. It is a stand-alone application that greatly simplifies its creation.

COMBINING THE THREE

We can skip that middle step of downloading tweets from Twitter's XML REST Web service, and download them directly from SAS, and have SXLE process the data with our map. And do it all in three lines of code:

```
filename sxlemap 'c:\tweets.map';
filename tweetlib url
'https://api.twitter.com/1/statuses/user_timeline.xml?screen_name=philihp';
libname tweetlib xml xmlmap=sxlemap access=readonly;
```

PARAMETERS AND FILTERING

This can be taken further by adding URL parameters directly into the URL string from macro variables. The Twitter API allows for an optional “exclude_replies=true” parameter which causes the output not to return any tweets in reply to other users. The screen name could as well be a macro variable.

```
%let user=philihp;
%let filtered=true;
filename sxlemap 'c:\tweets.map';
filename tweetlib url
```

```
          "https://api.twitter.com/1/statuses/
                      user_timeline.xml?screen_name=&user.%str(&)exclude_replies=&filtered";
libname tweetlib xml xmlmap=sxlemap access=readonly;
```

When including multiple parameters into the URL that will be passed to the GET HTTP request, it is important to escape the ampersand by surrounding it with %str(). This prevents SAS from treating it as a Macro variable.

CONCLUSION

Three lines of code! And your data is securely moved across the Internet and into your SAS session as a data set ready for you to work your analytic magic on it. Almost every social media Web site on the Internet is now a dynamic AJAX-driven Web site, and they all have APIs, most often REST APIs. By combining the FILENAME statement URL access method and the XML LIBNAME engine, XML from a RESTful Web service can be processed quickly by SAS with an incredibly minimal amount of code.

REFERENCES

- Richardson, Leonard & Sam Ruby. 2007. *RESTful Web Services*. Sebastopol, CA: O'Reilly Media, Inc.
- Helf, Garth. 2005. "Extreme Web Access: What to Do When FILENAME URL Is Not Enough." Proceedings of SUGI 30. Available at http://www2.sas.com/proceedings/sugi30/100-30.pdf
- Friebel, Anthony. 2002. "<XML> at SAS – A More Capable XML LIBNAME Engine." Proceedings of SUGI 27. Available at http://www2.sas.com/proceedings/sugi27/p179-27.pdf

CONTACT INFORMATION

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

Philip James Busby
SAS Campus Dr
SAS Institute, Inc.
philip.busby@sas.com