Paper 1646-2014

Hash It Out with a Web Service and Get the Data You Need

John Vickery, North Carolina State University

ABSTRACT

Have you ever needed additional data that was only accessible via a web service in XML or JSON? In some situations, the web service is set up to only accept parameter values that return data for a single observation. To get the data for multiple values, we need to iteratively pass the parameter values to the web service in order to build the necessary dataset. This paper shows how to combine the SAS® hash object with the FILEVAR= option to iteratively pass a parameter value to a web service and input the resulting JSON or XML formatted data.

INTRODUCTION

This paper describes how you can iteratively pass parameters to a web service in order to prepare a dataset by combining the SAS hash object and the FILEVAR= option. In our example, we start with a dataset of book ISBNs and we need to add author and title information. To add author and title data, we will make calls to the WorldCat xISBN web service. Documentation about this public web service is here: http://xisbn.worldcat.org/xisbnadmin/doc/api.htm.

SAS has many ways of accessing data from the web. For example, PROC HTTP, PROC SOAP, and the XML LIBNAME engine. We could also, of course, use the SET statement with BY group processing to iteratively pass values into the FILEVAR= variable. There are definitely situations where the methods mentioned above would be a better fit. For example, just using the SET and BY statements or a macro program with PROC HTTP could accomplish the same task presented in this paper.

The method described in this paper is offered as an alternative to the above methods and as an example of the utility of the hash object. For example, by using a hash object, the lookup dataset does not need to be pre-sorted. This method can be useful in the case where you need to perform a lookup function in order to fill in a dataset and the additional data is only made available via a web service. In many cases, these web services are designed to display information about a single item on a web page (with web applications in mind). They may not be designed for bulk data transfer. In order to get the data we need, we need to iteratively pass parameters to the web service through the URL.

SAMPLE DATASET AND OUTPUT

We start with a small sample dataset of ISBNs. Our task is to add title and author data to the ISBNS.

```
/* sample isbns */
data isbns;
   length isbn $ 13;
   input isbn;
datalines;
9781439867976
9781118147597
9781607649915
9781607646648
9781612903330
;
run;
```

The final output will include the ISBNs from the dataset created above as well as title and author. Output 1 shows the output from a PROC PRINT of the final dataset with author and title data.

Obs	author	title	isbn
1	Frank Ohlhorst	Big data analytics : turning big data into big money	9781118147597
2	Geoff Der, Brian S. Everitt	Applied medical statistics using SAS	9781439867976

Obs	author	title	isbn
3	Maura E. Stokes, Charles S. Davis, Gary G. Koch	Categorical data analysis using SAS	9781607646648
4	Art Carpenter	Carpenter's guide to innovative SAS techniques	9781607649915
5	Ron Cody	Cody's collection of popular SAS programming tasks and how to tackle them	9781612903330

Output 1. Output from a PROC PRINT of final dataset

WEB SERVICES

A web service is basically a method for communicating between devices over the web. The W3C defines a web service as, "software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." (http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice)

SENDING A REQUEST TO THE WEB SERVICE

In order to get data from the WorldCat xISBN web service, we need to send an HTTP request by passing URL parameters to the service. In particular, the service requires that a single ISBN be embedded in the URL. The request URL is of the general form:

`http://xisbn.worldcat.org/webservices/xid/isbn/<ISBN>?<p1=value>&<p2=value>&<pN=value>`

<*ISBN*> is the ISBN that we are requesting author and title data for and <*p1*>…<*pN*> are the parameter name and value pairs which specify the calling method and the data we are requesting and in what format. In this paper we include the following four parameters in the URL. Note that these parameters are specific to the WorldCat xISBN web service. Another web service would use different parameters.

1. ISBN
2. method=getMetadata
3. fl=author,title
4. format=json OR format=xml

URL request for JSON data:

http://xisbn.worldcat.org/webservices/xid/isbn/9781607646648?method=getMetadata&fl=author,title&format=json

URL request for XML data:

http://xisbn.worldcat.org/webservices/xid/isbn/9781607646648?method=getMetadata&fl=author,title&format=xml

In order to mask special characters in the URL components, the URL request is broken into parts and assigned to macro variables using the %NRSTR function. The current value of ISBN that is being passed to the web service is then concatenated with the two macro variables. The %SUPERQ function puts the values of the macro variables in quotes. According to the SAS documentation, "%SUPERQ is particularly useful for masking macro variables that might contain an ampersand or a percent sign."

```
%let baseurl = %nrstr(http://xisbn.worldcat.org/webservices/xid/isbn/);
%let params = %nrstr(?method=getMetadata&fl=author,title&format=json);  ❶

target = "%superq(baseurl)"||isbn||"%superq(params)";  ❷
```

1. Change the end of the PARAMS macro variable to `format=xml` to request XML data.

2. The `target` variable will be discussed in more detail in the following sections. Note that the `isbn` variable is concatenated between the two macro variables. This is the ISBN for which we are requesting author and title data.

For a more detailed discussion of the HTTP protocol and web service requests, see Busby (2012) and Helf (2005).

WEB SERVICE RESPONSE

Data from a call to a web service is typically returned in XML or JSON format. XML is probably familiar to most SAS programmers. An excellent explanation of XML can be found by Schacherer (2013). While SAS has the XML LIBNAME engine and the XML Mapper to access and parse XML data, an equivalent option is not available for JSON formatted data. In our example, we will know the structure of the web service return so we can use the DATA step to input and parse the data.

XML

Display 1 is an example XML response from the WorldCat xISBN web service. While it has been enlarged in the screen capture, note that there are five lines of data or records returned from the web service.

```
<?xml version="1.0" encoding="UTF-8"?>
<rsp xmlns="http://worldcat.org/xid/isbn/" stat="ok">
      <isbn   title="Categorical data analysis using SAS"  author="Maura E.
Stokes, Charles S. Davis, Gary G. Koch."     >9781607646648</isbn>

</rsp>
```

Display 1. An XML response from the WorldCat xISBN web service

JSON

JSON stands for JavaScript Object Notation. As described by Wikipedia, JSON "is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML." (http://en.wikipedia.org/wiki/JSON)

Display 2 is the same data as above but returned from the WorldCat xISBN web service in JSON format. Again, while it has been enlarged in the screen capture, note that with the JSON return, there are six lines of data or records returned from the web service.

```
{
 "stat":"ok",
 "list":[{
        "isbn":["9781607646648"],
        "author":"Maura E. Stokes, Charles S. Davis, Gary G. Koch.",
        "title":"Categorical data analysis using SAS"}]}
```

Display 2. A JSON response from the WorldCat xISBN web service

It is important to know how the particular web service you are calling will return the data. For example, it may be well structured with line breaks as in our example. Or, the web service may return the data as a continuous string as in Display 3. This will affect the ease with which you can parse the data.

```
{"stat":"ok","list":[{"isbn":["9781607646648"],"author":"Maura E. Stokes,
Charles S. Davis, Gary G. Koch.","title":"Categorical data analysis using
SAS"}]}
```

Display 3. A JSON response shown as a continuous string

The responsiveness of the web service should also be taken into account. PROC HTTP may be faster if you need to make changes to the default request header.

SAS® HASH OBJECT

The SAS hash object allows you to load a dataset into memory and perform lookup operations. With the hash iterator

object, we are able to traverse the hash object in ascending or descending order. The example in this paper makes use of the hash iterator object to iteratively pass each ISBN in the original dataset to the WorldCat xISBN web service.

Here we quickly review a few of the main points of the SAS hash object. Several excellent and detailed discussions are available. To really get up to speed on the SAS hash object, see for example, Eberhardt (2012), Dorfman (2010) or Burlew (2012).

A hash object is a component object that resides in memory and only exists during execution of the DATA step in which it was defined. A component object is accessible through the DATA Step Component Interface or DSCI and consists of attributes, methods and operators.

OBJECT ORIENTED TERMINOLOGY

When working with the SAS hash object, we use SAS statements to create and manipulate *component objects*.

Component objects are data elements that consist of attributes, methods, and operators. The example in this paper only makes use of two component objects – the hash object and the hash iterator object.

Attributes are the properties of a component object.

Methods are the operations that an object can perform. Our example makes use of the following methods: DEFINEKEY, DEFINEDATA, DEFINEDONE, FIRST and NEXT.

Operators provide special functionality.

The SAS hash object uses dot notation or object-dot-syntax. With this notation, you write the name of the object followed by a dot [.] and then the name of the method or attribute. For example, to use the *DEFINEDONE* method on a hash object named *MYHASH*, you would write the following:

```
myhash.definedone()
```

Optional arguments to the method are enclosed in the parentheses (). For example, to use the *DEFINEKEY* method, we would use the name of the key variable(s) as arguments as follows. Note that the argument is a string enclosed in quotes rather than the variable itself.

```
myhash.definekey('isbn')
```

DECLARING THE HASH

Our first step in using the ISBNs as a hash lookup table is to name and instantiate the hash object. We do so using the DECLARE statement.

```
data books;
   length author $ 100 title $ 250; ❷
   if _n_ = 0 then set isbns; ❸
   declare hash hisbn(dataset: 'isbns', ordered:'A'); ❶
… more code …
```

In this snippet:

1. Here we name and instantiate a hash object called **hisbn**. We also make use of the **dataset:** argument tag to load an existing dataset into the hash. This saves us from having to use the **add()** method to load the hash. We are telling SAS to create an in-memory object from the ISBNs dataset. Note that we use character strings instead of dataset names. While not shown here, regular SAS dataset options are available to us at this stage. For example, we could have written the following:

   ```
   dataset: 'isbns (where=(isbn not is null))'
   ```

   The `ordered:'A'` argument tag and option specifies that the data be returned in ascending key-value order when using the hash iterator object or the OUTPUT method. You can specify ascending, descending or leave the default of some undefined order.

   Another important aspect of instantiating a hash object is how it will handle duplicate or multiple key items. In our example, we do not have duplicate key values but I encourage the reader to refer to the documentation on the `MULTIDATA:` and `DUPLICATE:` argument tags.

2. The length statement here assigns the correct attributes for the data that we will retrieve from the WorldCat xISBN web service.

3. In the following section we discuss defining the key and data elements of the hash object. It is our responsibility to let the DATA know the correct attributes of these elements. By using a SET statement with the IF _N_ = 0 the data step will automatically read the attributes of the elements we are using in the HISBN hash. In this example, it is only the single variable ISBN.

DEFINING THE KEY AND THE DATA

Next, we need to tell SAS which variables will serve as the key and data elements of the hash. In our case, the key and data elements are the same.

```
data books;
   length author $ 100 title $ 250;
   if _n_ = 0 then set isbns;
   declare hash hisbn(dataset: 'isbns', ordered:'A');
   rc = hisbn.definekey('isbn');    ❶
❹  rc = hisbn.definedata('isbn');   ❷
   rc = hisbn.definedone();         ❸
… more code …
```

In this snippet:

1. First, the **`definekey()`** method is used to specify ISBN as the key for the **`hisbn`** hash. As with the **`dataset:`** argument tag used above with the DECLARE statement, the name of the key variable is a string rather than the variable itself.

2. Next, the **`definedata()`** method is used to specify ISBN as a data element for the **`hisbn`** hash. Note that we have to also include ISBN in the **`definedata()`** method in order for it to be written out to our final dataset.

3. The **definedone()** method is used to indicate that all key and data definitions for the **HISBN** hash object are complete.

4. We are also capturing the return code. All methods return a code specifies whether the method succeeded or failed. A return code of zero indicates success; a nonzero value indicates failure.

THE HASH ITERATOR

The hash iterator object allows us to retrieve the hash object data in ascending or descending order. In our example, we are making use of the iterator to step through each data value in the hash and use that as a parameter in our call to the web service.

To create a hash iterator object, we again use the DECLARE statement to name and instantiate the iterator object. The hash object must be declared and instantiated prior to creating an iterator object.

```
data books;
   length author $ 100 title $ 250;
   if _n_ = 0 then set isbns;
   declare hash hisbn(dataset: 'isbns', ordered:'A');
   rc = hisbn.definekey('isbn');
   rc = hisbn.definedata('isbn');
   rc = hisbn.definedone();
   declare hiter myiter('hisbn');   ❶
   rc = myiter.first();             ❷
… more code …
```

In this snippet:

1. Here the DECLARE statement is used to declare and instantiate the hash iterator object named MYITER. The previously created hash HISBN is referenced as a string in parentheses.

2. Here, the **first()** method is used with the MITER iterator object. Note that the name of the iterator precedes the method. The **first()** method will return the first data item in the HISBN hash object. Because we specified

the `ordered:'A'` argument tag in the DECLARE statement when the hash was created, the method will return the data item with the 'least' key (i.e. smallest numeric value or first alphabetic character).

Looking forward, we will use the **next()** method with the MYITER iterator object to iteratively traverse the hash object and return the data items in ascending key order.

DYNAMICALLY CHANGING THE INPUT FILE

As we traverse the hash object in order to pass an ISBN to the WorldCat xISBN web service, we need a way to dynamically close the input file and open a new one. To do so, we use the FILEVAR= option on the INFILE statement. In this case, the input file that we are changing is the request to the web service. For each key value in the hash object, we will pass a new ISBN parameter to the web service via the URL.

FILEVAR= OPTION

According to the SAS documentation, the FILEVAR= option "specifies a variable whose change in value causes the INFILE statement to close the current input file and open a new one. When the next INPUT statement executes, it reads from the new file that the FILEVAR= variable specifies. Like automatic variables, this variable is not written to the data set."

In our example, the input file that we are changing is the target web service URL.

```
%let baseurl = %nrstr(http://xisbn.worldcat.org/webservices/xid/isbn/);   ❶
%let params = %nrstr(?method=getMetadata&fl=author,title&format=json);

data books;
   length author $ 100 title $ 250;
   if _n_ = 0 then set isbns;
   declare hash hisbn(dataset: 'isbns', ordered:'A');
   rc = hisbn.definekey('isbn');
   rc = hisbn.definedata('isbn');
   rc = hisbn.definedone();
   declare hiter myiter('hisbn');
   rc = myiter.first();
   do while (rc=0);    ❷
❸     target = "%superq(baseurl)"||isbn||"%superq(params)";
      infile wcws url filevar=target lrecl=4000 scanover debug;   ❹
  … more code …
```

In this snippet:

1. The macro variables discussed in the "Web Services" section are added to build the components of the request URL.

2. Here, the DO WHILE loop is based on the value of the return code which was set by the **first()** method. At the bottom of the loop, we will use the **next()** method to reset the return code.

3. The variable `target` is the URL that we pass to the web service. Note that we concatenate the current value of `isbn` in the correct location in the URL. Each time the hash iterator moves to the next key, a new value of ISBN is passed into the URL. This URL is then passed to the WorldCat xISBN web service. Note also that the FILEVAR=*variable* is not written to the output dataset.

4. On the INFILE statement:

   a. `wcws` is the *file-specification*. When using the FILEVAR= option, it is a placeholder rather than an actual filename or a previously assigned fileref. The placeholder is used for reporting to the log and must conform to the same rules as a fileref.
   b. `url` specifies the URL access method.
   c. `filevar=target` specifies the variable whose change in value causes the INFILE statement to close the current input file and open a new one.
   d. `scanover` causes the INPUT statement to scan the input data records until the character string that is specified in the @'*character-string*' expression is found. We will look at the INPUT statement more closely in the next section.

e. `debug` writes information to log. We can use this to see the HTTP headers that the server returns.

PARSING THE JSON OR XML RETURNED FROM THE WEB SERVICE

Because we are using the URL access method on the INFILE statement, we are not making use of an XML map for the XML returned data. Similarly, if we request JSON data to be returned from the web service, SAS cannot automatically parse the structure. We can use prior knowledge of how the web service will return data to construct our INPUT statement to extract the data we need.

THE @'CHARACTER-STRING' COLUMN POINTER

To extract the title and author data associated with each ISBN that we pass to the web service, we will use the *@'character-string'* column pointer control to move the pointer to the first column of data we need to input. Specifically, the *@'character-string'* column pointer "locates the specified series of characters in the input record and moves the pointer to the first column after *character-string*."

Note that we must list the character strings in the order in which they are located in the returned data.

By testing a response from the web service, we know that it will return XML data in the format shown in Display 1. Likewise, a request for JSON data will be returned as shown in Display 2. In the following snippet, we are requesting JSON formatted data.

```
%let baseurl = %nrstr(http://xisbn.worldcat.org/webservices/xid/isbn/);
%let params = %nrstr(?method=getMetadata&fl=author,title&format=json);

data books;
   length author $ 100 title $ 250;
   if _n_ = 0 then set isbns;
   declare hash hisbn(dataset: 'isbns', ordered:'A');
   rc = hisbn.definekey('isbn');
   rc = hisbn.definedata('isbn');
   rc = hisbn.definedone();
   declare hiter myiter('hisbn');
   rc = myiter.first();
   do while (rc=0);
      target = "%superq(baseurl)"||isbn||"%superq(params)";
      infile wcws url filevar=target lrecl=4000 scanover debug;
      input @'"author":"' author $ &    ❶
            @'"title":"' title $ & ;
      author = substr(author,1,index(author,'."')-1);   ❷
      title = substr(title,1,index(title,'"}')-1);
… more code …
```

In this snippet:

1. Here, we use the *@'character-string'* column pointer on the INPUT statement. From Display 2, we know that the author data will follow the string `"author":"`. Likewise, title data will follow the string `"title":"`.
   a. These strings correspond to the JSON name/value pair structure.
   b. We also make use of the "&" format modifier in order to read in the data containing embedded blanks. According to the SAS documentation, "[t]he & (ampersand) format modifier enables you to read character values that contains one or more embedded blanks with list input and to specify a character informat. SAS reads until it encounters two consecutive blanks, the defined length of the variable, or the end of the input line, whichever comes first."
2. We still need to remove the trailing JSON syntax, so we use the SUBSTR function.

In the snippet above, we requested JSON formatted data by specifying "format=json" in the URL. Changing that to "format=xml" would have resulted in a XML data being returned as shown in Display 1. To parse an XML response, we could make the following changes to the INPUT statement.

```
Input @'title=' title $ &
      @'author=' author $ & ;
```

Note that the WorldCat xISBN web service places title before author in an XML formatted response. And again, the SUBSTR and other character functions can be used to remove XML syntax.

STEP-BY-STEP THROUGH THE CODE

This section puts together the pieces discussed above and looks at the complete program line by line. Shown first is the code to request and parse a JSON response. Then, minor changes are made to request and parse an XML response from the web service.

JSON RESPONSE

```
     /* JSON version */
1.   %let baseurl = %nrstr(http://xisbn.worldcat.org/webservices/xid/isbn/);
2.   %let params = %nrstr(?method=getMetadata&fl=author,title&format=json);

3.   data books(drop=rc);
4.      length author $ 100 title $ 250;
5.      if _n_ = 0 then set isbns;
6.      declare hash hisbn(dataset: 'isbns', ordered:'A');
7.      rc = hisbn.definekey('isbn');
8.      rc = hisbn.definedata('isbn');
9.      rc = hisbn.definedone();
10.     declare hiter myiter('hisbn');
11.     rc = myiter.first();
12.     do while (rc=0);
13.        target = "%superq(baseurl)"||isbn||"%superq(params)";
14.        infile wcws url filevar=target lrecl=4000 truncover scanover debug;
15.        input @'"author":"' author $ &
                 @'"title":"' title $ & ;
16.        author = substr(author,1,index(author,'."')-1);
17.        title = substr(title,1,index(title,'"}')-1);
18.        output;
19.        rc = myiter.next();
20.     end;
21.     stop;
22.  run;
```

1. Assign the base portion of the request URL to macro variable and use the %NRSTR function to mask special characters.
2. Assign the parameter portion of the request URL to macro variable and use the %NRSTR function to mask special characters. Note that the value for the FORMAT parameter is JSON.
3. Name the new dataset and drop the return code variable.
4. Create variables for the author and title data that will be returned from the WorldCat xISBN web service.
5. The SAS compiler reads the column names and attributes from the ISBNS dataset and adds the column(s) to the program data vector (PDV). The statement is not executed as _N_ will never equal zero (0) but by doing so, we do not have to explicitly let the DATA step know the attributes of the key and data element of the hash. In this case, there is only the single ISBN variable in the dataset.
6. Using the DECLARE statement we name and instantiate the `hisbn` hash object. The `dataset: 'isbns'` argument tag specifies that we are using the ISBNS dataset to load into the hash object. Note that the dataset name is in quotes. The `ordered:'A'` argument tag specifies that the data in the hash be returned in ascending key-value order.
7. The **definekey()** method is invoked on the `hisbn` hash object to specify the ISBN variable as the key. Again, the variable name must be enclosed in quotes. The `RC` variable captures the return code from the **definekey()** method. A value of zero indicates success; a nonzero value indicates failure.
8. The **definedata()** method is invoked on the `hisbn` hash object to specify the ISBN variable as a data element in the hash. Although ISBN is also defined as a key variable, it must still be listed here as we need the hash object to return the key value when the hash item is accessed. Again, the variable name must be enclosed in quotes and the `RC` variable operates the same as with the **definekey()** method.
9. The **definedone()** method is called to indicate that all key and data definitions are complete.

10. The DECLARE statement names and instantiates a hash iterator object (`hiter`). The iterator object is named `myiter`. The argument is the name of the hash object enclosed in quotes `('hisbn')`.

11. The **first()** method is called on the `myiter` iterator object. This method will return the first data item in the `hisbn` hash object. Since we used the `ordered:'A'` argument tag in the DECLARE statement in line 4, the data item that is returned is the one with the 'least' key (i.e. smallest numeric value or first alphabetic character). The `RC` variable is again capturing the return code of the method.

12. Begin a DO LOOP to iterate as long as the value of the `RC` variable is zero indicating a successful call to a hash object method. For the first iteration of the loop, the value of the RC variable is based on the call to the **first()** method on the iterator object. At the bottom of the loop, we reset the value of RC based on a call to the **next()** method.

13. The variable `target` is the URL that we pass to the web service. Note that we concatenate the current value of `isbn` in the correct location in the URL. Each time the hash object iterates, a new value of ISBN is passed into the URL. This URL is then passed to the WorldCat xISBN web service. Note also that the FILEVAR=*variable* is not written to the output dataset.

14. On the INFILE statement:

    a. `wcws` is the *file-specification*. When using the FILEVAR= option, it is a placeholder rather than an actual filename or a previously assigned fileref. The placeholder is used for reporting to the log and must conform to the same rules as a fileref.
    b. `url` specifies the URL access method.
    c. `filevar=target` specifies the variable whose change in value causes the INFILE statement to close the current input file and open a new one.
    d. `scanover` causes the INPUT statement to scan the input data records until the character string that is specified in the @'character-string' expression is found. We will look at the INPUT statement more closely in the next section.
    e. `debug` writes information to log. We can use this to see the HTTP headers that the server returns.

15. The INPUT statement uses the @'*character-string*' column pointer to locate the author and title data in the JSON formatted return from the web service. The & (ampersand) format modifier is enables the INPUT statement to read character values that contain one or more embedded blanks.

16. Use the SUBSTR function to remove the trailing JSON syntax characters from the `author` variable.

17. Use the SUBSTR function to remove the trailing JSON syntax characters from the `title` variable.

18. Write the current observation out to the BOOKS dataset.

19. A call to the **next()** method on the `myiter` iterator object returns the next ISBN value in the hash. It also resets the value of the `rc` variable to either 1 or 0. After the last value of ISBN is returned, the **next()** method the `rc` variable will be set to 1 (i.e. failure) and SAS will exit the loop.

20. Just the corresponding END statement for the DO LOOP.

21. The STOP statement prevents an infinite loop.

22. RUN!

XML RESPONSE

To request an XML response from the web service, only four lines from the above code need to be modified.

In line #11, the FORMAT parameter in the URL must be changed to `format=xml`.

```
2. %let params = %nrstr(?method=getMetadata&fl=author,title&format=xml);
```

In line #15, TITLE must precede AUTHOR and the format of the @'*character-string*' column pointer is slightly different.

```
15.  input @'title=' title $ &
           @'author=' author $ & ;
```

In lines #14 and #15, the COMPRESS function is used rather than the SUBSTR function in order to remove unwanted quote characters present in the XML formatted data.

```
16. title = compress(title,'"');
17. author = compress(author,'"');
```

CONCLUSION

This paper was intended to show how to combine the SAS hash object and the FILEVAR= option to iteratively pass parameters to a web service and input the resulting JSON or XML response. SAS has many tools to access data on the web such as the PROC HTTP and the XML LIBNAME engine as well as methods to iteratively change the input file. However, the method described in this paper offers an option that may prove useful as you encounter data only made available via a web service.

REFERENCES

Burlew, M. 2012. *SAS Hash Object Programming Made Easy.* Cary NC: SAS Institute.

Busby, Philip. 2012. "A Simple Way of Importing from a REST Web Service into SAS in Three Lines of Code." *Proceedings of the SAS Global Forum 2012 Conference.*
http://support.sas.com/resources/papers/proceedings12/075-2012.pdf

Dorfman, Paul M., Vyverman, Koen, and Dorfman, Victor P. 2010. "Black Belt Hashigana." *Proceedings of the SAS Global Forum 2010 Conference.*
http://support.sas.com/resources/papers/proceedings10/023-2010.pdf

Eberhardt, Peter. 2012. "The SAS Hash Object: It's Time to .find() Your Way Around." *Proceedings of the SAS Global Forum 2012 Conference.*
http://support.sas.com/resources/papers/proceedings12/147-2012.pdf

Helf, Garth. 2005. "Extreme Web Access: What to Do When FILENAME URL Is Not Enough". *Proceedings of the SUGI 2005 Conference.*
http://www2.sas.com/proceedings/sugi30/100-30.pdf

Schacherer, Chris. 2013. "The SAS Programmer's Guide to XML and Web Services." *Proceedings of the SAS Global Forum 2013 Conference.*
http://support.sas.com/resources/papers/proceedings13/124-2013.pdf

RECOMMENDED READING

- Dunn, Toby and Lafler, Kirk Paul. 2012. "The Good, The Bad, and The Ugly." *Proceedings of the SAS Global Forum 2013 Conference.* http://support.sas.com/resources/papers/proceedings12/226-2012.pdf
- Cody, Ronald. 2004. "The INPUT Statement: Where It's @." *Proceedings of the SUGI 2004 Conference.* http://www2.sas.com/proceedings/sugi29/253-29.pdf
- First, Steven. 2008. "The SAS INFILE and FILE Statements". *Proceedings of the SAS Global Forum 2008 Conference.* http://www2.sas.com/proceedings/forum2008/166-2008.pdf
- McNeill, Bill. 2013. "The Ins and Outs of Web-Based Data with SAS." *Proceedings of the SAS Global Forum 2013 Conference.* http://support.sas.com/resources/papers/proceedings13/024-2013.pdf

CONTACT INFORMATION

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

Name:	John Vickery
Organization:	NCSU Libraries, North Carolina State University
Address:	Box 7111
City, State ZIP:	Raleigh, NC 27695
Work Phone:	(919) 513-0344
Email:	john_vickery@ncsu.edu