Paper SAS2024-2018

Accessing Microservice Data in SAS® 9.4 with DS2 Built-in Packages

James P. Kelley, SAS Institute Inc., Cary, NC

ABSTRACT

This paper describes how, as of SAS® 9.4M3, to use the DS2 built-in packages for HTTP get and post operations, JSON parsing, and hash mapping to access data from a restful microservice. The example code details how to get an HTTP response from a service, parse the JSON returned, normalize the values into data rows, and then use the SAS® DS2 hash package to write those rows into a data set for further processing.

INTRODUCTION

It is sometimes useful to be able to save the response data into a data set that can be probed in a name-value pair style rather than using a record-based mode. For example, you might wish to contact a microservice and retrieve data from it to use as part of a DS2 workflow while using a generic way to pre-process that data from JSON into name-value pairs. This can be a challenge if the service does not confirm to a strict schema, or it might return variants for objects.

To allow the user to form a result data set in a well-organized and modular way, this paper details the functionality according to the different stages of processing (input, traversal, and output) using user-declared packages to extend and enhance the functionality of the HTTP, JSON, and hash packages provided as DS2 built-ins.

PACKAGE DESIGN IN DS2

To process and access the microservices data that we will use, you use a user-defined package. User-defined packages are available in DS2 as a way of encapsulating data and adding functionality much in the same way other languages use objects. Packages can contain properties, methods, and constructors, and they can reference other packages. Here is the initial structure of a user-defined package:

```
proc ds2;
  package JsonParser / overwrite=yes[1];
      dcl package[2] json    tokenizer();
      dcl package  logger[3] putlog();
      dcl int              rc;

      method JsonParser(in_out varchar[4] text);
         rc = tokenizer.createParser( text );
            …
      end;

      method NextEntry(…) returns int;
         …
         return rc;
      end;

      method DELETE()[5];
         rc = tokenizer.destroyParser();
         if ( rc ne 0 ) then
            do;
            putlog.log('E', 'JsonParser: tokenizer deletion failed.');
                  || 'Could not destroy the instance of JSON package. Rc='
                  || rc || '.');
            STOP[6];
            end;
```

```
    end;
endpackage;
```

Here are the elements of note in the example above:

1. Use of `/ overwrite=yes` ensures that changes in the package declaration are respected. Packages declarations are persistent, so pre-existing declarations are retained between sessions. Use of the overwrite option ensures that the declaration is refreshed.
2. Packages can encapsulate instances of other packages. For example, `dcl package json tokenizer()` will create an instance of the DS2 JSON built-in package.
3. Outputting messages to the log can be done by declaring a `package logger` instance and calling the `log(…)` method on it.
4. The use of an `IN_OUT` with `varchar` will remove the need to specify the length of the string. But, it will no longer accept quoted string constants as a parameter.
5. Any package that has internally declared packages should contain a way of disposing of those instances when they are no longer needed. Here, that method is called `DELETE()` to mirror the standard method name for deleting built-in package instances.
6. `STOP` will work to stop the current DS2 program even from inside a method in a user-declared package.

ACCESSING THE MICROSERVICE

Mark Jordan (also known as SAS Jedi) wrote an excellent and detailed summary of how to use the HTTP package in his post, “Jedi SAS Tricks: DS2 & APIs – GET the data that you are looking for” in the SAS Training blog [1]. He used the SCAN and SUBSTR functions to parse the replies, but we will be using the JSON package to parse the entire reply.

In Appendix A, there is the user-defined package MicroServiceAPI, which is a distilled implementation of Mark’s paper. It will access a microservice and do a simple GET call for a response with optional accept, contentType, and authorization headers.

An example of how to access the National Institute of Health’s RXNAV[2] microservice API[3] and retrieve a response with the concepts dictionary using this user-defined package would be as follows:

```
dcl varchar(29381061) character set utf8 response;
dcl varchar(32767) character set utf8 headers text contentType;
dcl integer rc statusCode;
dcl double elapsed;
dcl double last;

microservice.setAccept('application/json; charset=utf-8;');
microservice.setContentType('application/json; charset=utf-8;');
rc=microservice.get('https://rxnav.nlm.nih.gov/REST/allconcepts?tty=ALL',
                    statusCode, headers, contentType, response);
```

USING THE DS2 JSON PARSING PACKAGE

The JSON package that is provided as a built-in with DS2 is a tokenizing JSON parser. Chris Hemedinger posted a write-up (“Using SAS DS2 to parse JSON” on The SAS Dummy blog [4]) on how to use it to parse data if you know the schema beforehand in a JSON response.

The initial user-defined JsonParser package is extended to wrap the JSON built-in parser. Then, the built-in parser does the tokenizing and the user-defined package uses a state machine to build object and list representations of the data. In order to do this, several other user-defined packages need to be created to help manage the data.

WRAPPING THE DATA TO BE MANAGED

Appendix B contains JsonPathElement, which is the first user-defined package needed. It manages and encapsulates the data necessary to create a single item in a flattened linked list of JSON path elements and the value for each element.

These elements are updated multiple times as the state machine processes the tokens. Their values are reliable only after the entirety of the JSON response is processed successfully and returned by a call to the NextEntry() method of the user-defined package JsonParser.

MANAGING THE LIST OF THE WRAPPED DATA

Appendix C contains JsonPathList, which is the second user-defined package used to manage data. It is for the holder of the head of that list and controls the management of elements in it. This is a very standard double-linked list management utility and can be easily reused by changing the element type as needed.

USING THE JSON PARSER TO POPULATE THE LIST

Appendix D contains JsonParser, which has been updated from the initial user-defined package to include a state machine base that uses the built-in parser's tokens to populate and traverse that list.

The built-in parser reads tokens and returns their parsed values, along with a token type ID of `LeftBracket`, `LeftBrace`, `RightBracket`, `RightBrace`, `Label`, `Null`, `String`, `Float`, `Integer`, `BooleanFalse`, `BooleanTrue`, and `Numeric`.

The user-defined package JsonParser interprets these as array start, object start, array end, object end, label, and the value types string, float, integer, Boolean, and numeric. It updates the list of elements to store the values.

USING THE DS2 HASH PACKAGE

The final stage is for the hash package to make a key from the path to each key-value or list-value and the values returned when that pair was parsed. This allows for JSON-path style addressing in lookups.

To do this, the variables to be processed have to be in the data vector and declared at the start:

```
proc ds2;
   data _null_;/* using hash table output, so use _null_ here */
      dcl package Hash            outputHash();
      /* variables to go into the output hash */
      dcl int                                  id;
      dcl varchar(14)    character set utf8    elementType;
      dcl varchar(32767) character set utf8    elementPath;
      dcl varchar(32767) character set utf8    elementName;
      dcl int                                  arrayIndex;
      dcl varchar(32767) character set utf8    fullpath;
      dcl varchar(32767) character set utf8    valueType;
      dcl varchar(32767) character set utf8    value;
      dcl varchar(32767) character set utf8    state;
      dcl varchar(32767) character set utf8    tokenType;
      dcl varchar(32767) character set utf8    tokenValue;
```

The hash object then needs to be initialized as follows:

```
   method init();

      /* define output hash key and data which becomes
         the subsequent dataset columns */
      outputHash.defineKey('id');
      outputHash.ordered('ASCENDING');
      outputHash.defineData('id');
```

```
        * outputHash.defineData('state');
        outputHash.defineData('fullpath');
        outputHash.defineData('elementPath');
        outputHash.defineData('elementName');
        outputHash.defineData('arrayIndex');
        outputHash.defineData('elementType');
        outputHash.defineData('valuetype');
        outputHash.defineData('value');
        * outputHash.defineData('tokenType');
        * outputHash.defineData('tokenValue');
        outputHash.defineDone();

    end;
```

Populating the parser list after getting the response from the microservice as shown above is done as follows:

```
parser = _NEW_ JsonParser(response);
```

Once the hash object has its data elements defined, the hash object must be populated from the processed list of parsed elements. The method ParseToHash in Appendix E does that.

```
rc = ParseToHash(parser,outputHash);
```

And, after calling the ParseToHash method to load the parsed data into the hash object, the output method on the hash method must be invoked.

```
rc = outputHash.output('RXNAV_ALL_CONCEPTS');
```

THE FINAL OUTPUT TABLE

The final output table now contains a list of all the properties and values in the JSON broken out by pathname-value pairs. The layout below can be used to construct queries to perform operations on that data that would be cumbersome in its original JSON layout.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
5	arrayIndex	Num	8
4	elementName	Char	32767
3	elementPath	Char	32767
6	elementType	Char	14
2	fullpath	Char	32767
1	id	Num	8
8	value	Char	32767
7	valueType	Char	32767

PUTTING IT ALL TOGETHER

A full practical example shown in Appendix E uses the Yahoo Weather API[5] to build a weather forecast for the Denver, CO area. It uses a query string of `select * from weather.forecast where woeid in (select woeid from geo.places(1) where text="denver, co")`.

```
microservice.setAccept('application/json; charset=utf-8;');
microservice.setContentType('application/json; charset=utf-8;');
/* QUERY: select * from weather.forecast where woeid
          in (select woeid from geo.places(1) where text="denver, co") */
 rc = microservice.get('https://query.yahooapis.com/v1/public/yql'
                     || '?q=select%20*%20from%20weather.forecast%20'
                     || 'where%20woeid%20in%20(select%20woeid%20'
                     || 'from%20geo.places(1)%20where%20text%3D%22'
                     || 'denver%2C%20co%22)&format=json&env=store'
                     || '%3A%2F%2Fdatatables.org%2'
                     || 'Falltableswithkey',
                    statusCode, headers, contentType, response);
```

The response this returns varies by the day it is run, but today, it reads in part:


```
{"query":{... "results":{"channel":{ ... "title":"Yahoo! Weather - Denver, CO, US", ... "item":{ ...
"forecast":[{"code":"23","date":"05 Mar 2018","day":"Mon","high":"45","low":"29","text":"Breezy"},
{"code":"32","date":"06 Mar 2018","day":"Tue","high":"46","low":"24","text":"Sunny"},
{"code":"32","date":"07 Mar 2018","day":"Wed","high":"53","low":"20","text":"Sunny"},
{"code":"30","date":"08 Mar 2018","day":"Thu","high":"64","low":"27","text":"Partly Cloudy"},
{"code":"30","date":"09 Mar 2018","day":"Fri","high":"63","low":"29","text":"Partly Cloudy"},
{"code":"30","date":"10 Mar 2018","day":"Sat","high":"57","low":"33","text":"Partly Cloudy"},
{"code":"30","date":"11 Mar 2018","day":"Sun","high":"60","low":"31","text":"Partly Cloudy"},
{"code":"30","date":"12 Mar 2018","day":"Mon","high":"64","low":"34","text":"Partly Cloudy"},
{"code":"30","date":"13 Mar 2018","day":"Tue","high":"64","low":"32","text":"Partly Cloudy"},
{"code":"34","date":"14 Mar 2018","day":"Wed","high":"67","low":"33","text":"Mostly Sunny"}],
...
```


The parse for this generates a table of entries that includes the full path to each node, the partial path to that node (elementPath), the name of that node (elementName), its index if it is an element of an array or if it is not, the type of node that it is, the type of the value assigned to that node, and the text of that value:

id	fullpath	elementPath	elementName	arrayIndex	elementType	valueType	value
111	/query/results/channel/item/forecast/0/	/query/results/channel/item/forecast/0	/	0	Array/Object Pair	Properties	{...}
120	/query/results/channel/item/forecast/0/low	/query/results/channel/item/forecast/0/	low	.	Property/Value Pair	string	28
114	/query/results/channel/item/forecast/0/date	/query/results/channel/item/forecast/0/	date	.	Property/Value Pair	string	05 Mar 2018
122	/query/results/channel/item/forecast/0/text	/query/results/channel/item/forecast/0/	text	.	Property/Value Pair	string	Breezy
118	/query/results/channel/item/forecast/0/high	/query/results/channel/item/forecast/0/	high	.	Property/Value Pair	string	41
112	/query/results/channel/item/forecast/0/code	/query/results/channel/item/forecast/0/	code	.	Property/Value Pair	string	23
116	/query/results/channel/item/forecast/0/day	/query/results/channel/item/forecast/0/	day	.	Property/Value Pair	string	Mon
125	/query/results/channel/item/forecast/1/	/query/results/channel/item/forecast/1	/	1	Array/Object Pair	Properties	{...}
134	/query/results/channel/item/forecast/1/low	/query/results/channel/item/forecast/1/	low	.	Property/Value Pair	string	24
128	/query/results/channel/item/forecast/1/date	/query/results/channel/item/forecast/1/	date	.	Property/Value Pair	string	06 Mar 2018
136	/query/results/channel/item/forecast/1/text	/query/results/channel/item/forecast/1/	text	.	Property/Value Pair	string	Sunny
132	/query/results/channel/item/forecast/1/high	/query/results/channel/item/forecast/1/	high	.	Property/Value Pair	string	46
126	/query/results/channel/item/forecast/1/code	/query/results/channel/item/forecast/1/	code	.	Property/Value Pair	string	32
130	/query/results/channel/item/forecast/1/day	/query/results/channel/item/forecast/1/	day	.	Property/Value Pair	string	Tue
139	/query/results/channel/item/forecast/2/	/query/results/channel/item/forecast/2	/	2	Array/Object Pair	Properties	{...}

To finish out the example, the query to build a rowwise summary of conditions by day would be:

```
select max(date) as date, max(day) as day, max(low) as low,
       max(high) as high, max(text) as text, elementPath
   from (
      select value as date, '', '', '', '', elementPath
        from YW_DENVER_CO
        where elementName='date' and elementType='Property/Value'
          and elementPath contains '/query/results/channel/item/forecast'
       union select '', value as day,  '', '', '', elementPath
        from YW_DENVER_CO
        where elementName='day' and elementType='Property/Value'
          and elementPath contains '/query/results/channel/item/forecast'
       union select '', '', value as low,  '', '', elementPath
        from YW_DENVER_CO
        where elementName='low' and elementType='Property/Value'
          and elementPath contains '/query/results/channel/item/forecast'
       union select '', '', '', value as high, '', elementPath
        from YW_DENVER_CO
        where elementName='high' and elementType='Property/Value'
          and elementPath contains '/query/results/channel/item/forecast'
       union select '', '', '', '', value as text, elementPath
        from YW_DENVER_CO
        where elementName='text' and elementType='Property/Value'
          and elementPath contains '/query/results/channel/item/forecast'
       ) group by elementPath order by elementPath
```

And, it would return the following results:

date	day	low	high	text	elementPath
05 Mar 2018	Mon	29	45	Breezy	/query/results/channel/item/forecast/0/
06 Mar 2018	Tue	24	46	Sunny	/query/results/channel/item/forecast/1/
07 Mar 2018	Wed	20	53	Sunny	/query/results/channel/item/forecast/2/
08 Mar 2018	Thu	27	64	Partly Cloudy	/query/results/channel/item/forecast/3/
09 Mar 2018	Fri	29	63	Partly Cloudy	/query/results/channel/item/forecast/4/
10 Mar 2018	Sat	33	57	Partly Cloudy	/query/results/channel/item/forecast/5/
11 Mar 2018	Sun	31	60	Partly Cloudy	/query/results/channel/item/forecast/6/
12 Mar 2018	Mon	34	64	Partly Cloudy	/query/results/channel/item/forecast/7/
13 Mar 2018	Tue	32	64	Partly Cloudy	/query/results/channel/item/forecast/8/
14 Mar 2018	Wed	33	67	Mostly Sunny	/query/results/channel/item/forecast/9/

CONCLUSION

DS2 user-declared packages can simplify and create generic solutions that can be leveraged to solve much more complicated data processing needs.

NOTES

[1] Jordan, Mark (also known as SAS Jedi). 2015. "Jedi SAS Tricks: DS2 & APIs - GET the data you are looking for." SAS Learning Blog. Posted January 17, 2015. Available

http://blogs.sas.com/content/sastraining/2015/01/17/jedi-sas-tricks-ds2-apis-get-the-data-you-are-looking-for

[2] RxNav 2.0. 2018. U.S. National Library of Medicine (NLM). National Institutes of Health, Department of Health and Human Services. Available https://rxnav.nlm.nih.gov/. Accessed March 2, 2018.

[3] RxNav 2.0 Rest API. 2018. Ibid. Available https://rxnav.nlm.nih.gov/RxNormAPIs.html. Accessed March 2, 2018.

[4] Hemedinger, Chris. 2015. "Using SAS DS2 to parse JSON." The SAS Dummy Blog. Posted September 28, 2015. Available https://blogs.sas.com/content/sasdummy/2015/09/28/parse-json-from-sas/.

[5] Yahoo Weather API 2008. Oath Holdings Inc. Available https://developer.yahoo.com/weather/. Accessed March 2, 2018.

APPENDIX A: DS2_PACKAGE_MICROSERVICEAPI

```
proc ds2;
package MicroServiceAPI / overwrite = yes;
   dcl package logger putlog();
   dcl package http web;
   dcl varchar(32767) authorization;
   dcl varchar(32767) accept;
   dcl varchar(32767) contentType;

   method setAuthorization(varchar(32767) value);
      authorization=value;
   end;

   method setAccept       (varchar(32767) value);
      accept=value;
   end;

   method setContentType  (varchar(32767) value);
      contentType=value;
   end;

   method MicroServiceAPI();
      authorization = null; accept = null; contentType  = null;
   end;

   method get(varchar(32767) url, IN_OUT integer statusCode,
              IN_OUT varchar headers, IN_OUT varchar responseContentType,
              IN_OUT varchar response) returns bigint;
      dcl varchar(32767) character set utf8 text;
      dcl integer j rc;

      web = _NEW_ http();
      web.createGetMethod(url);
      if (not null(accept))
         then web.addRequestHeader('Accept',accept);
      if ( not null(contentType) )
         then web.addRequestHeader('Content-Type',contentType);
      if ( not null(authorization) )
          then web.addRequestHeader('Authorization',authorization);

      rc = web.executeMethod();
      statusCode = web.getStatusCode();
```

```
      if (rc ne 0) then do; goto done; end;
      if (statusCode ne 200) then do; rc = -1; goto done; end;

      web.getResponseHeadersAsString(headers, rc);
      if (rc ne 0) then do; goto done; end;

      web.getResponseContentType(responseContentType,rc);
      if (rc ne 0) then do; goto done; end;

      web.getResponseBodyAsString(response, rc);
      stop:
      web.DELETE();
      return rc;
   end;
endpackage;
run;
quit;
```

APPENDIX B: DS2_PACKAGE_JSONPATHELEMENT

```
package JsonPathElement / overwrite=yes;
   dcl package JsonPathElement prev;
   dcl package JsonPathElement next;
   dcl int                                 id;
   dcl varchar(32767) character set utf8 name;
   dcl varchar(14)    character set utf8 type;
   dcl int                                 arrayIndex;
   dcl varchar(32767) character set utf8 state;

   method JsonPathElement(int id1,
                          varchar(32767) character set utf8 name1,
                          varchar(32767) character set utf8 state1,
                          varchar(64)    character set utf8 type1,
                          int arrayIndex1
                          );
      id = id1; name = name1; state = state1; type = type1;
      arrayIndex = arrayIndex1;
   end;

   method toString() returns varchar(32767) character set utf8;
      return '[id=' || id || ' name="' || name ||
             '" arrayIndex=' || arrayIndex || ' type="' || type ||
             '" state="' || state || '"]';
   end;
endpackage;
```

APPENDIX C: DS2_PACKAGE_JSONPATHLIST

```
package JsonPathList / overwrite=yes;
   dcl package logger putlog();
   dcl package JsonPathElement head;

   method JsonPathList();
      head = null;  * null is an empty list;
   end;

   method GetHead() returns package JsonPathElement;
```

```
   return head;
end;

method size() returns int;
   dcl int count;
   dcl package JsonPathElement step;
   if ( null(head) ) then return 0;
   count = 1;
   step = head.next;
   do while ( ^null(step) );
      step = step.next;
      count = count+1;
   end;
   return count;
end;

/* find the first element of a chain of elements. */
method first( package JsonPathElement element )
              returns package JsonPathElement;
   dcl package JsonPathElement step;
   if ( null(element) ) then return element;
      step = element;
      do while ( ^null(step.prev) );
         step = step.prev;
   end;
   return step;
end;

/* find the last element of a chain of elements. */
method last( package JsonPathElement element )
             returns package JsonPathElement;
   dcl package JsonPathElement step;
   if ( null(element) ) then return element;
   step = element;
   do while ( ^null(step.next) );
      step = step.next;
   end;
   return step;
end;

/* add an element (or a chain of elements) into the head
   of the list and relink the list to the tail of that. */
method push( package JsonPathElement element );
   dcl package JsonPathElement newHead;
   dcl package JsonPathElement join;
   newHead = first(element);
   if ( ^null(head) ) then do;
      join = last(element);
      head.prev = join;
      join.next = head;
      end;
   head = newHead;
end;

/* add an element (or a chain of elements) onto the tail
   of the list and return the last element in the entire list. */
method append( package JsonPathElement element )
```

```
                returns package JsonPathElement;
   dcl package JsonPathElement joinFrom;
   dcl package JsonPathElement joinTo;
   joinTo = first(element);
   if ( null(head) ) then do;
      head = joinTo;
      end;
   else do;
      joinFrom = last(head);
      joinFrom.next = joinTo;
      joinTo.prev = joinFrom;
      end;
   return last(element);
end;

/* get the element at the index and return it without unlinking it. */
method get( int index ) returns package JsonPathElement;
   dcl package JsonPathElement element;
   if ( index eq 0 ) then do;
      element = head;
      end;
   else if ( index < 0 ) then do;
      dcl int inc;
      inc = index + 1;
      element = last(head);
      do while ( inc ne 0 );
         if ( null(element.prev) ) then do;
            /* ERROR: out of range. */
            putlog.log('E', 'Index ' || index || ' is out of range.');
            STOP;
            end;
         element = element.prev;
         inc = inc + 1;
         end;
      end;
   else do;
      dcl int inc;
      inc = 0;
      element = head;
      do while ( inc ne index );
         if ( null(element.next) ) then do;
            /* ERROR: out of range. */
            putlog.log('E', 'Index ' || index || ' is out of range.');
            STOP;
            end;
         element = element.prev;
         inc = inc + 1;
         end;
      end;
   return element;
end;

/* Relinks two element chains so that the second chain is
 * now the successor of the element pointed to by the first
 * chain and the rest of the elements after the first element
 * in the first chain are appended to the end of the second chain. */
method relink( package JsonPathElement joinFrom,
```

```
                package JsonPathElement joinTo );
   /*              joinFrom                        joinTo             */
   /*  BEFORE  (a<--->_b_<--->(c...))   ((x...)<--->_y_<--->(z...))  */
   /*  AFTER   (a<--->_b_<--->(x...)<--->_y_<--->(z...)<--->(c...)   */
   /*  change           ^3  ^2                          ^4  ^1        */
   dcl package JsonPathElement reparentNext;
   dcl package JsonPathElement reparentHead;
   dcl package JsonPathElement reparentTail;
   reparentNext = joinFrom.next;  /* (c...) */
   reparentHead = first(joinTo);  /* (x...) */
   reparentTail = last(joinTo);   /* (z...) */
   if ( ^null(reparentNext) ) then do;
     reparentNext.prev = reparentTail; /* 1 */
     end;
   if ( ^null(reparentTail) ) then do;
     reparentTail.next = reparentNext; /* 4 */
     end;
   joinFrom.next = reparentHead;         /* 2 */
   if ( ^null(reparentHead) ) then do;
     reparentHead.prev = joinFrom;     /* 3 */
     end;
end;

/* inserts the element at the index specified. */
method insert( int index, package JsonPathElement element );
   if ( index eq 0 ) then do;
      if ( null(head) ) then do;
         head = first(element);
         end;
      else do;
         dcl package JsonPathElement joinFrom;
         joinFrom = last(element);
         relink(joinFrom,head);
         end;
      head = first(element);
      end;
   else do;
      if ( null(head) ) then do;
         /* ERROR: empty list. */
         putlog.log('E', 'Cannot insert at index '
                         || index || ' of an empty list.');
         STOP;
         end;
      else do;
         dcl package JsonPathElement joinFrom;
         joinFrom = get(index);
         relink( joinFrom, element );
         end;
      end;
end;

/* return the element at the head of the list, unlink it,
   and relink the list. */
method pop() returns package JsonPathElement;
   dcl package JsonPathElement element;
   element = head;
   head = head.next;
```

```
      if ( ^null(head.next) ) then do;
         head.prev = null;
         end;
      element.next = null;
      element.prev = null;
      return element;
   end;

   /* return the element at the index given (zero based) and
      unlink it, and relink the list. */
   method removeAt( int index ) returns package JsonPathElement;
      dcl package JsonPathElement element;
      if ( null(head) ) then do;
         putlog.log('E', 'Cannot delete from an empty list.');
         STOP;
         end;
      else if ( index eq 0 | (null(head.next) & null(head.prev))) then do;
         element = head;
         head = head.next;
         if ( ^null(head) ) then do;
            head.prev = null;
            end;
         element.next = null;
         element.prev = null;
         end;
      else do;
         element = get(index);
         relink( element.prev, element.next );
         element.next = null;
         element.prev = null;
         end;
      return element;
   end;
endpackage;
```

APPENDIX D: DS2_PACKAGE_JSONPARSER

```
package JsonParser / overwrite=yes;
   dcl package logger       putlog();
   dcl package json         tokenizer();
   dcl package JsonPathList pathList();
   /* keep track of the original text and its length. */
   dcl int                                    inputLength;
   dcl varchar(10000000) character set utf8 inputText;
   /* variables during parsing. */
   dcl int                                   rc;
   dcl int                                   tokenTypeId;
   dcl int                                   parseFlags;
   dcl bigint                                lineNum;
   dcl bigint                                colNum;
   dcl varchar(32767)  character set utf8  token;
   dcl int                                   runningId;

   method JsonParser(in_out varchar text);
      rc = tokenizer.createParser( text );
      if ( rc ne 0 ) then do;
         putlog.log('N', 'JsonParser: tokenizer creation rc = '||rc );
```

```
      putlog.log('E', 'JsonParser: tokenizer creation failed.  '
       || 'Could not create an JSON package for the provided text.');
      STOP;
      end;
   inputText=text;
   inputLength=length(text);
   parseFlags = 0;
   lineNum = 0;
   colNum = 0;
   token = null;
   runningId = 0;
end;

method DELETE();
   rc = tokenizer.destroyParser();
   if ( rc ne 0 ) then do;
      putlog.log('N', 'JsonParser: lineNum=' || lineNum
                   || ' colNum=' || colNum || ' token=' || token
                   || ' tokenType=' || tokenTypeId
                   || ' parseFlags=' || parseFlags || '.');
      putlog.log('N', 'JsonParser: tokenizer deletion rc='||rc||'.');
      putlog.log('E', 'JsonParser: tokenizer deletion failed.  '
              || 'Could not destroy the instance of JSON package.');
      STOP;
   end;
end;

method toString() returns varchar(100000000) character set utf8;
   return '[JsonParser: rc=' || rc || ' tokenType=' || tokenTypeId
          ||' parseFlags=' || parseFlags ||' lineNum=' || lineNum
          ||' colNum=' || colNum || ' token="' || token
          ||' inputLength=' || inputLength
          || '" inputText="' ||inputText || '"]';
end;

/* HasMoreTokens() can be called to see if the stream has more tokens
   to process, but NextEntry() may still return null after
   processing them. */
method HasMoreTokens() returns int;
   if (colNum lt inputLength) then return 1;
   return 0;
end;

/* STOP - the parse of the text ended while there were still
   partial objects, arrays, or properties on the stack. */
method STOP_Unexpected_EOI();
   putlog.log('E', 'JsonParser failed.  End of input with non-empty'
              || ' parse stack: ' || pathList.toString() || '.');
   putlog.log('N', 'JsonParser: input = ['||inputText||']');
   STOP;
end;

/* STOP - the parse of the text did not yield a token when expected. */
method STOP_Unexpected_NoToken();
   putlog.log('N', 'JsonParser: lineNum=' || lineNum
                || ' colNum=' || colNum || ' token=' || token
                || ' tokenType=' || tokenTypeId
```

```
                || ' parseFlags=' || parseFlags || '.');
   putlog.log('N', 'JsonParser: next token rc = '||rc||'.');
   putlog.log('E', 'JsonParser: get next token failed.  '
                || 'Could not retrieve next token.');
   putlog.log('N', 'JsonParser: input = "'||inputText||'"');
   STOP;
end;

/* STOP - the token encountered is not consistant with this
   program's supported JSON syntax. */
method STOP_Invalid_Type();
   putlog.log('N', 'JsonParser: lineNum=' || lineNum
                || ' colNum=' || colNum || ' token=' || token
                || ' tokenType=' || tokenTypeId
                || ' parseFlags=' || parseFlags || '.');
   putlog.log('E', 'JsonParser: parsed token type invalid.  '
                || 'Only one of isNull='
                || tokenizer.isNull(tokenTypeId)
                || ' isString=' || tokenizer.isString(tokenTypeId)
                || ' isNumeric=' || tokenizer.isNumeric(tokenTypeId)
                || ' isFalse=' || tokenizer.isBooleanFalse(tokenTypeId)
                || ' isTrue='  || tokenizer.isBooleanTrue(tokenTypeId)
                || ' is allowed to be non-zero.');
   STOP;
end;

/* STOP - the token encountered is not consistant with this
   program's supported JSON syntax. */
method STOP_Invalid_Token();
   putlog.log('N', 'JsonParser: lineNum=' || lineNum
                || ' colNum=' || colNum || ' token=' || token
                || ' tokenType=' || tokenTypeId
                || ' parseFlags=' || parseFlags || '.');
   putlog.log('E', 'JsonParser failed.  Only one of isArrStart='
          || tokenizer.isLeftBracket(tokenTypeId)
          || ' isArrEnd=' || tokenizer.isRightBracket(tokenTypeId)
          || ' isObjStart=' || tokenizer.isLeftBrace(tokenTypeId)
          || ' isObjEnd=' || tokenizer.isRightBrace(tokenTypeId)
          || ' isLabel=' || tokenizer.isLabel(tokenTypeId, parseFlags)
          || ' is allowed to be non-zero.');
   STOP;
end;

method deleteElement(package JsonPathElement element);
   if ( ^null(element) ) then do;
      putlog.log('N','delete '||element.toString());
      element.id = -1 * element.id;
      element.DELETE();
      end;
end;

method GetTokenType(in_out varchar tokenType);
   /* sanity check parsed token */
   if ( (tokenizer.isNull(tokenTypeId)
        + tokenizer.isString(tokenTypeId)
        + tokenizer.isNumeric(tokenTypeId)
        + tokenizer.isBooleanFalse(tokenTypeId)
```

```
        + tokenizer.isBooleanTrue(tokenTypeId)) gt 1 )
        then STOP_Invalid_Type();

    if ( (tokenizer.isLeftBracket(tokenTypeId)
        + tokenizer.isLeftBrace(tokenTypeId)
        + tokenizer.isRightBracket(tokenTypeId)
        + tokenizer.isRightBrace(tokenTypeId)
        + tokenizer.isLabel(tokenTypeId, parseFlags)) gt 1 )
        then STOP_Invalid_Token();

    /* structure tokens */
    if ( tokenizer.isLeftBracket(tokenTypeId) ne 0 ) then
       tokenType='Array';
    else if ( tokenizer.isLeftBrace(tokenTypeId) ne 0 ) then
       tokenType='Object';
    else if ( tokenizer.isRightBracket(tokenTypeId) ne 0 ) then
       tokenType='ArrayEnd';
    else if ( tokenizer.isRightBrace(tokenTypeId) ne 0 ) then
       tokenType='ObjectEnd';
    else if ( tokenizer.isLabel(tokenTypeId, parseFlags) ne 0 ) then
       tokenType='Label';
    /* value tokens */
    else if ( tokenizer.isNull(tokenTypeId) ne 0 ) then
       tokenType='null';
    else if ( tokenizer.isString(tokenTypeId) ne 0 ) then
       tokenType='string';
    else if ( tokenizer.isFloat(tokenTypeId, parseFlags) ne 0 ) then
       tokenType='float';
    else if ( tokenizer.isInteger(tokenTypeId, parseFlags) ne 0 ) then
       tokenType='integer';
    else if ( tokenizer.isBooleanFalse(tokenTypeId) ne 0 ) then
       tokenType='boolean';
    else if ( tokenizer.isBooleanTrue(tokenTypeId) ne 0 ) then
       tokenType='boolean';
    else if ( tokenizer.isNumeric(tokenTypeId) ne 0 ) then
       tokenType='numeric';
    else
       tokenTypeId='-?-'; /* unknown token */
       /* tokenizer.isPartial(parseFlags) not handled in this example */
end;

method buildPath(in_out varchar elementPath);
  dcl package JsonPathElement step;
  elementPath = '';
  if ( not null(pathList.head) ) then do;
     step = pathList.GetHead();
     do while ( not null(step) );
        if ( step.arrayIndex ge 0 ) then do;
           elementPath = elementPath || '/' || step.arrayIndex;
           end;
        else do;
           elementPath = elementPath || step.name;
           end;
        step = step.next;
        end;
     end;
  else do;
```

```
      elementPath = '';
      end;
end;

method getArrayIndex(in_out int arrayIndex);
   dcl package JsonPathElement currentElement;
   arrayIndex = .;
   if ( not null(pathList.getHead()) ) then do;
      currentElement = pathList.get(-1);
      if ( currentElement.type eq 'Array' ) then do;
         arrayIndex = currentElement.arrayIndex;
         end;
      end;
end;

/* NextEntry() will return the next start of an object, an array, or
   a property value. Parsing is only completely done when NextEntry()
   returns null. */
method NextEntry(in_out int     id,
                 in_out varchar elementType,
                 in_out varchar elementPath,
                 in_out int     arrayindex,
                 in_out varchar elementName,
                 in_out varchar fullPath,
                 in_out varchar valueType,
                 in_out varchar valueText,
                 in_out varchar state,
                 in_out varchar tokenType,
                 in_out varchar tokenValue
                 ) returns int;
   dcl int keepGoing;
   dcl package JsonPathElement step;
   elementType = null;
   elementName = null;
   elementPath = null;
   fullPath = null;
   valueText = null;
   valueType = null;
   state = null;
   arrayindex = .;
   /* id - is a running count, do not clear */
   /* only return an entry for the actual properties, objects,
      and arrays. */
   keepGoing = 1;
   do while ( keepGoing ne 0 );
      valueText = '';
      rc = 0;
      if ( colNum ge inputLength ) then do;
         putlog.log('N','End of Text.');
         elementPath = '/';
         arrayIndex = .;
         elementType = null;
         state = null;
         valueText = null;
         keepGoing = 0;
         /* (eof)- exit loop */
         end;
```

```
else do;
   tokenizer.getNextToken( rc, token, tokenTypeId,
                           parseFlags, lineNum, colNum );
   if ( ( rc ne 0 ) and ( rc ne 101 ) ) then do;
      STOP_Unexpected_NoToken();
      end;
   tokenValue = token;
   GetTokenType(tokenType);
   valueType = .;
   /* S -> {L:S, ... } | [S, ... ] | V */
   keepGoing = 0;
   if ( tokenType eq 'Label' ) then do;
      /* L : Label (Property Start) */
      dcl package JsonPathElement labelElement;
      /* build path before adding element */
      buildPath(elementPath);
      getArrayIndex(arrayIndex);
      runningId = runningId + 1;
      id = runningId; /* hash to new entry */
      arrayIndex = .;
      state = 'inLabel';
      elementName = token;
      elementType = 'Property';
      valueType = 'Value';
      labelElement =_NEW_ JsonPathElement(id,elementName,
                                          state,elementType, . );
      pathList.append(labelElement);
      /* go get the next value as the value of the property */
      end;
   else if ( tokenType eq 'ObjectEnd' ) then do;
      /* } : Object End */
      dcl package JsonPathElement currentElement;
      runningId = runningId + 1;
      currentElement = pathList.get(-1);
      id = currentElement.id; /* hash to original entry */
      elementName = currentElement.name;
      valueType = 'Properties';
      pathList.removeAt(-1);
      /*     { "x":1 } = obj1, prop, value, obj1end
                       : pop just returned the obj1.
             [ { } ]   = arr, obj1, obj1end
                       : pop just returned the obj1.
             { "x":{} } = obj1, prop, obj2, obj2end
                       : pop just returned obj2
       */
      if ( not null(pathList.getHead()) ) then do;
         currentElement = pathList.get(-1);
         if ( currentElement.type eq 'Array' ) then do;
            elementType = 'Array/Object Pair';
            valueText = '{...}';
            state = 'paired';
            /* build path after removing element */
            buildPath(elementPath);
            getArrayIndex(arrayIndex);
            end;
         else if ( currentElement.type eq 'Property' ) then do;
            elementType = 'Property/Object Pair';
```

```
            valueText = '{...}';
            state = 'paired';
            pathList.removeAt(-1);
            /* build path after removing element */
            buildPath(elementPath);
            getArrayIndex(arrayIndex);
            elementName = currentElement.name;
            id = currentElement.id; /* hash to original entry */
            end;
         else do;
            elementType = '?? object end ??';
            id = runningId; /* hash to new entry */
            valueText = token;
            state = '??';
            pathList.removeAt(-1);
            /* build path after removing element */
            buildPath(elementPath);
            getArrayIndex(arrayIndex);
            end;
         end;
      else do;
         elementType = 'Object';
         valueText = '{...}';
         state = 'top';
         /* build path after removing element */
         buildPath(elementPath);
         getArrayIndex(arrayIndex);
         end;
      end;
   else if ( tokenType eq 'ArrayEnd' ) then do;
      /* ] : Array End */
      dcl package JsonPathElement currentElement;
      runningId = runningId + 1;
      currentElement = pathList.get(-1);
      id = currentElement.id; /* hash to original entry */
      elementName = currentElement.name;
      valueType = 'Elements';
      pathList.removeAt(-1);
      if ( not null(pathList.getHead()) ) then do;
         currentElement = pathList.get(-1);
         if ( currentElement.type eq 'Array' ) then do;
            elementType = 'Array/Array Pair';
            valueText = '[...]';
            state = 'paired';
            end;
         else if ( currentElement.type eq 'Property' ) then do;
            elementType = 'Property/Array Pair';
            valueText = '[...]';
            pathList.removeAt(-1);
            state = 'paired';
            end;
         else do;
            elementType = '?? array end ??';
            id = runningId; /* hash to new entry */
            valueText = token;
            pathList.removeAt(-1);
            state = '??';
```

```
            end;
         end;
      else do;
         elementType = 'Array';
         valueText = '[...]';
         state = 'top';
         end;
      buildPath(elementPath);
      getArrayIndex(arrayIndex);
      end;
   else if ( tokenType eq 'Object' ) then do;
      /* { : Object Start */
      dcl package JsonPathElement objectElement;
      dcl package JsonPathElement currentElement;
      if ( not null(pathList.getHead()) ) then do;
         currentElement = pathList.get(-1);
         if ( currentElement.type eq 'Array' ) then do;
            currentElement.arrayIndex =
                        currentElement.arrayIndex + 1;
            end;
         end;
      /* build path before adding element */
      buildPath(elementPath);
      getArrayIndex(arrayIndex);
      runningId = runningId + 1;
      elementName = '/';
      id = runningId; /* hash to new entry */
      state = 'inObject';
      elementType = 'Object';
      valueType = 'Properties';
      objectElement =_NEW_ JsonPathElement(id,elementName,
                                  state,elementType, . );
      pathList.append(objectElement);
      end;
   else if ( tokenType eq 'Array' ) then do;
      /* [ : Array Start */
      dcl package JsonPathElement currentElement;
      dcl package JsonPathElement arrayElement;
      if ( not null(pathList.getHead()) ) then do;
         currentElement = pathList.get(-1);
         if ( currentElement.type eq 'Array' ) then do;
            currentElement.arrayIndex =
                     currentElement.arrayIndex + 1;
            end;
         end;
      /* build path before adding element */
      buildPath(elementPath);
      getArrayIndex(arrayIndex);
      runningId = runningId + 1;
      id = runningId;    /* hash to new entry */
      elementName = '';
      state = 'inArray';
      elementType = 'Array';
      valueType = 'Elements';
      arrayElement =_NEW_ JsonPathElement(id,elementName,
                                   state,elementType, -1 );
      pathList.append(arrayElement);
```

```
            /* return the array's info ahead of its contents */
            end;
         else do;
            /* V : Value - can occur in an array, a property,
                           or as the top level value. */
            dcl package JsonPathElement currentElement;
            runningId = runningId + 1;
            if ( not null(pathList.getHead()) ) then do;
               currentElement = pathList.get(-1);
               elementName = currentElement.name;
               valueText = token;
               valueType = tokenType;
               if ( currentElement.type eq 'Array' ) then do;
                  currentElement.arrayIndex =
                           currentElement.arrayIndex + 1;
                  elementType = 'Array/Value Pair';
                  id = runningId; /* hash to new entry */
                  state = 'paired';
                  end;
               else if ( currentElement.type eq 'Property' ) then do;
                  elementType = 'Property/Value Pair';
                  id = currentElement.id; /* hash to original entry */
                  state = 'paired';
                  pathList.removeAt(-1);
                  end;
               else do;
                  elementType = '?? value ??';
                  state = '??';
                  id = runningId; /* hash to new entry */
                  pathList.removeAt(-1);
                  end;
            end;
         else do;
            elementType = 'Value';
            valueText = token;
            id = runningId; /* hash to new entry */
            state = 'top';
            end;
         /* build path after removing element */
         buildPath(elementPath);
         getArrayIndex(arrayIndex);
         keepGoing = 0;
         /* return the value */
         end;
      end;
   end;
   fullPath = elementPath;
   if ( not null(elementName) ) then do;
      fullPath = fullPath || elementName;
      end;
   return 0;  /* rc = 0 (ok) */
end;
endpackage;
```

APPENDIX E: AN EXAMPLE

```
%INCLUDE 'src\DS2_Package_JsonPathElement.sas';
```

```
%INCLUDE 'src\DS2_Package_JsonPathList.sas';
%INCLUDE 'src\DS2_Package_JsonParser.sas';
%INCLUDE 'src\DS2_Package_MicroServiceAPI.sas';

OPTIONS SOURCE SOURCE2 DS2SCOND=ERROR ERRORCHECK=STRICT;

proc datasets nolist; delete 'YW_DENVER_CO'n; run; quit;
proc ds2;
   data _null_;
      dcl package logger          putlog();
      dcl package Hash            outputHash();
      dcl package MicroServiceAPI microservice();
      dcl package JsonParser      parser;
      dcl int                 rc;
      /* variables to go into the output hash */
      dcl int                                  id;
      dcl varchar(14)    character set utf8    elementType;
      dcl varchar(32767) character set utf8    elementPath;
      dcl varchar(32767) character set utf8    elementName;
      dcl int                                  arrayIndex;
      dcl varchar(32767) character set utf8    fullpath;
      dcl varchar(32767) character set utf8    valueType;
      dcl varchar(32767) character set utf8    value;
      dcl varchar(32767) character set utf8    state;
      dcl varchar(32767) character set utf8    tokenType;
      dcl varchar(32767) character set utf8    tokenValue;

      Forward ParseToHash;

      method init();
         /* define output hash key and data which becomes the
              subsequent dataset columns */
         outputHash.defineKey('id');
         outputHash.ordered('ASCENDING');
         outputHash.defineData('id');
         * outputHash.defineData('state');
         outputHash.defineData('fullpath');
         outputHash.defineData('elementPath');
         outputHash.defineData('elementName');
         outputHash.defineData('arrayIndex');
         outputHash.defineData('elementType');
         outputHash.defineData('valuetype');
         outputHash.defineData('value');
         * outputHash.defineData('tokenType');
         * outputHash.defineData('tokenValue');
         outputHash.defineDone();
      end;

      method run();
         dcl varchar(32767) character set utf8 url;
         dcl varchar(29381061) character set utf8 response;
         dcl varchar(32767) character set utf8 headers text contentType;
         dcl integer i j rc statusCode;

         microservice.setAccept('application/json; charset=utf-8;');
         microservice.setContentType('application/json; charset=utf-8;');
         /* QUERY:
```

```
      select * from weather.forecast where woeid in
        (select woeid from geo.places(1) where text="denver, co") */
   rc = microservice.get('https://query.yahooapis.com/v1/public/yql'
                      || '?q=select%20*%20from%20weather.forecast%20'
                      || 'where%20woeid%20in%20('
                      || 'select%20woeid%20from%20geo.places(1)%20'
                      || 'where%20text%3D%22denver%2C%20co%22)&'
                      || 'format=json&'
                      || 'env=store%3A%2F%2Fdatatables.org%2'
                      || 'Falltableswithkeys',
                     statusCode, headers, contentType, response);
   if ( (rc ne 0) or (statusCode ne 200) ) then do;
      putlog.log('E', 'Web Query Failed: rc = '||rc ||
                       ', statusCode = '||statusCode||'.');
      STOP;
   end;

   /* microservice object no longer needed. */
   microservice.DELETE();
   microservice = null; /* let term() know it's been destroyed */

   /* start a new json parser based on that response */
   parser = _NEW_ JsonParser(response);
   /* use that json parser to populate a hash object */
   rc = ParseToHash(parser,outputHash);
   /* get rid of the parse, it's no longer needed and could
          have a lot of memory behind it now. */
   parser.DELETE();
   parser = null; /* let term() know it's been destroyed */
   /* output the hash from the json to a data set */
   rc = outputHash.output('YW_DENVER_CO');
end;

method term();
   if ( not null(microservice) ) then do;
      microservice.DELETE();
      end;
   if ( not null(parser) ) then do;
      parser.DELETE();
      end;
   outputHash.DELETE();
end;

/* Method to add all the entries to a hash indexed by both
   the unique id and the unique element path. */
method ParseToHash(package JsonParser parser,
                   package Hash outputHash) returns int;
   dcl package JsonPathElement element;
   dcl package JsonPathElement step;

   do while( parser.HasMoreTokens() );
      rc = parser.NextEntry(id, elementType, elementPath,
                         arrayIndex, elementName, fullPath,
                         valueType, value, state, tokenType,
                         tokenValue);
      if ( (rc ne 0 ) or null(value) ) then do;
         if ( null(value) ) then do;
```

```
                    rc = 0; /* end of parsing */
                    end;
                 goto done_parsing;
                 end;
              else do;
                 outputHash.replace();
                 end;
          end; /* while */
          done_parsing:
          return rc;
       end;
    enddata;
    run;
quit;

/* show the definition of the dataset created */
PROC CONTENTS data='YW_DENVER_CO'n; run;

/* show the count of the rows of the dataset created */
proc sql;
select * from YW_DENVER_CO where elementPath contains
      '/query/results/channel/item/forecast'
      order by elementPath;
quit;

/* show the count of the rows of the dataset created */
proc sql;
   select max(date) as date, max(day) as day, max(low) as low,
          max(high) as high, max(text) as text, elementPath
     from (
        select value as date, '', '', '', '', elementPath
          from YW_DENVER_CO
           where elementName='date' and elementType='Property/Value'
           and elementPath contains '/query/results/channel/item/forecast'
        union select '', value as day,  '', '', '', elementPath
         from YW_DENVER_CO
          where elementName='day' and elementType='Property/Value'
           and elementPath contains '/query/results/channel/item/forecast'
        union select '', '', value as low,  '', '', elementPath
         from YW_DENVER_CO
          where elementName='low' and elementType='Property/Value'
           and elementPath contains '/query/results/channel/item/forecast'
        union select '', '', '', value as high, '', elementPath
         from YW_DENVER_CO
          where elementName='high' and elementType='Property/Value'
          and elementPath contains '/query/results/channel/item/forecast'
        union select '', '', '', '', value as text, elementPath
         from YW_DENVER_CO
          where elementName='text' and elementType='Property/Value'
          and elementPath contains '/query/results/channel/item/forecast'
   ) group by elementPath order by elementPath;
quit;
```

ACKNOWLEDGMENTS

I’d like to thank Gloria Faley and David Shamlin, for their aid and support writing this paper; Robert Ray, Kat Schikore, and Bryan Ewbank of the DS2 team for their aid; Rick Langston for being an island of wisdom in the ocean of SAS; and Caroline Brickley along with Amy Wolfe for helping me with 11th hour edits.

RECOMMENDED READING

- SAS Institute Inc. 2017. *SAS 9.4 DS2 Language Reference,* 6th ed. Cary, NC: SAS Institute Inc. Available http://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.3&docsetId=ds2ref&docsetTarget=titlepage.htm&locale=en.

  See especially these chapters and examples:

  - "DS2 JSON Package Methods, Operators, and Statements"
  - "DS2 HTTP Package Methods, Operators, and Statements"
  - "DS2 Hash and Hash Iterator Package Attributes, Methods, Operators, and Statements"
  - Example: "Using an HTTP and JSON Package to Extract and Parse a REST-Formatted File"
  - Example: "Reading JSON Text from HTTP and Creating a SAS Data Set"

CONTACT INFORMATION

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

James P. Kelley
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
James.Kelley@sas.com
http://www.sas.com