## Paper 10981-2016

# Rapid Web Services using SAS/IntrNet® Software, iQuery AJAX, and PROC JSON

Jeremy Palbicki, Mayo Clinic

#### **ABSTRACT**

Creating web applications and web services can sometimes be a daunting task. With the ever changing facets of web programming, it can be even more challenging. For example, client-side web programming using applets was very popular almost 20 years ago, only to be replaced with server-side programming techniques. With all of the advancements in JavaScript libraries, it seems client-side programming is again making a comeback. Amidst all of the changing web technologies, surprisingly one of the most powerful tools I have found that has provided exceptional capabilities across all types of web development techniques has been SAS/IntrNet®. Traditionally seen as a server-side programming tool for generating complete web pages based on SAS® content, SAS/IntrNet, coupled with jQuery AJAX or AJAX alone, also has the ability to provide client-side web programming techniques, as well as provide RESTful web service implementations. I hope to show that with the combination of these tools including the JSON procedure from SAS® 9.4, simple yet powerful web services or dynamic content rich web pages can be created easily and rapidly.

## INTRODUCTION

SAS/IntrNet has been a tool of choice for developers for many years to provide server side data and or reports. However with more developers moving towards client side programming, we can't lose sight that SAS/IntrNet can still be used for providing developers with what they need most of all: data. SAS/IntrNet uses Common Gateway Interface (CGI) technology to provide access to SAS programs that generate an HTTP response, which in essence is really no different than a PHP, Perl or Java Servlet. It simply is a way to call a server side program and get a response. Adding SAS's JSON procedure to the mix, one can create RESTful compliant web services in a pinch. One can also create feature rich web applications using jQuery and AJAX, or any other JavaScript library available and utilize SAS data in real time. All that is needed is access to your SAS/IntrNet server, a few short lines of SAS code, and a web page to create your content.

#### **SETUP**

Before I get started, I will make the first assumption that the user has an understanding of HTML, JavaScript, jQuery, JSON, and some SAS. One does not need a lot of knowledge about each area but some will be required.

The first thing needed is to have your SAS/IntrNet server available. Contact your local SAS Administrator to find out where your server is and where you can place your SAS programs.

Next you will need access to a web page editor. Really any text editor will work. I use WordPad or Notepad++.

Finally if you are planning to use jQuery and jQuery AJAX, I recommend using the CDN (Content Delivery Network) for including the jQuery library. All you need to do is add this line into your HTML page:

#### <head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
</head>

## **CREATING THE WEB PAGE**

So the first part we need is to create the web page. In this example I will create a rather simple web page that uses an AJAX call to get data from a SAS program and returns the data to the web page.

Using the jQuery tutorial available from <a href="http://www.w3schools.com/jquery/">http://www.w3schools.com/jquery/</a> I was able to whip up this simple example:

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.3.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $.getJSON("http://sassrv/cgi-bin/broker? program=example.test.sas& service=default",
function(result){
                 $.each(result, function(i, field){
        $("div").append(field + " ");
      });
    });
  });
});
</script>
</head>
<body>
<button>Get JSON data</button>
<div></div>
</body>
</html>
```

You probably notice that the function I use is the jQuery function getJSON, which is where this gets powerful in the next section.

## **CREATING THE SAS PROGRAM**

As stated earlier, one of the functions I used in the jQuery example was getJSON. The reason for this is that with SAS 9.4, there is a new procedure called PROC JSON which produces a JSON Object or JSON Array. Typically this would be an output file than can be imported by other JSON tools. However when PROC JSON is used with the SAS/IntrNet server and the HTTP output file handle \_webout; one can return the JSON object or array directly to the calling routine.

Here is a very simple example of a SAS program that does this:

```
proc json out=_webout;
  export sashelp.class (where=(age=11)) / nokeys nosastags;
run;
```

In fact this is what is in the SAS/IntrNet program "test.sas" called by the previous HTML. On clicking the button from the web page, this data is added below it:

```
Joyce, F, 11, 51.3, 50.5 Thomas, M, 11, 57.5, 85
```

The above data generated is the JSON Array object from the SAS data set sashelp.class.

## **UNDER THE COVERS**

Clearly the above example showed that one can easily pull data from a SAS data set into a web page with just a few simple yet elegant lines of code. So what is going on? With the SAS/IntrNet tool, one can create web requests, such as GET, HEAD, POST, PUT etc... and as seen above with a simple GET request from the web page, the call to the cgi-bin/broker program tells the SAS/IntrNet server to execute the program and return the response. In the past, this usually meant that the SAS program had to generate a new page completely to display to the user, which meant programming numerous put statements to generate the HTML strings needed to represent the page. However with AJAX and jQuery, one only needs to return a simple string of text or in the above case a JSON object.

## **CREATING A RESTFUL WEBSERVICE**

What is a RESTFul web service? Oracle® defines RESTful web services this way:

"RESTful web services are built to work best on the Web. Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using **Uniform Resource Identifiers (URIs)**, typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol."

So to create a RESTful web service one has to provide a uniform interface, most likely using the traditional HTTP methods GET, POST, PUT, DELETE etc... and provide efficient, scalable operations and self-descriptive response messages. So let's examine the example from before. We provided a URI <a href="http://sassrv/cgi-bin/broker?">http://sassrv/cgi-bin/broker?</a> program=example.test.sas&\_service=default. By nature this is a GET request and the response it returned was a JSON Array. All three criteria I mentioned have been satisfied, so the above example is already an example of a RESTful web service.

# **ANOTHER EXAMPLE**

Building on my previous example which still uses the getJSON() function, but instead of using a hard coded "age=11" parameter in the SAS code in the where clause of the data set, I have now added a macro variable that is created via the get request:

```
...
<script>
$(document).ready(function(){
    $("button").click(function(){
     $.getJSON("http://sassrv/cgi-bin/broker?_program=example.test.sas &age=12",
function(result){
     $.each(result, function(i, field){
```

```
$("div").append(field + " ");
});
});
});
</script>
```

The new SAS code becomes:

```
proc json out=_webout;
  export sashelp.class (where=(age=&age)) / nokeys nosastags;
run;
```

This results in the following response on the web page:

```
James, M, 12, 57, 3, 83 Jane, F, 12, 59, 8, 84, 5 John, M, 12, 59, 99, 5 Louise, F, 12, 56, 3, 77 Robert, M, 12, 64, 8, 128
```

One can start to see the flexibility of these simple queries to SAS data, which can be retrieved easily using the web and RESTful web services.

## **PROC JSON**

The use of the getJSON method with jQuery AJAX was done on purpose to show off one of the new procedures with SAS 9.4: PROC JSON, a procedure that creates a JSON Object or JSON Array based on data from a SAS data set. In the past if you needed to create a JSON object based on SAS data either for a file or to display on the web as a response of a SAS/IntrNet call, one would have to craft the output to be compliant with the JSON structure using SAS put statements, a very tedious process. Now with PROC JSON, one only needs to export the data from the SAS data set into either a file or in our case the file handle webout which sends the JSON Object to the HTTP response.

Here is the definition of JSON from JSON.org:

"JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate...

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence

An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by: (colon) and the name/value pairs are separated by, (comma).

An *array* is an ordered collection of values. An array begins with [ (left bracket) and ends with ] (right bracket). Values are separated by , (comma).

A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.

A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.

A *number* is very much like a C or Java number, except that the octal and hexadecimal formats are not used."

Examples of simple JSON structures:

```
A simple JSON Object

{"firstname":"John", "lastname":"Doe"}

A simple JSON Array

{"employees":[
    {"firstname":"John", "lastname":"Doe"},
    {"firstname":"Jane", "lastname":"Doe"},
    {"firstname":"Jeremy", "lastname":"Palbicki"}
]}
```

## **COMPLETE EXAMPLE**

In the final example, one can see that you don't have to use the get method or the getJSON method each time. One can also use the POST, PUT, etc... methods for data interchange, making this process match the RESTful web service definition.

The following example uses the post() function of jQuery AJAX to insert data into a SAS data set. Below is the contents of the HTML file:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
<script>
$(document).ready(function(){
   $("button").click(function(){
     var prog = "example.test2.sas";
     var serv = "default";
     var last = ("#last").val();
     var first = $("#first").val();
        $.post("http://sassrv/cgi-bin/broker ", {_program: prog, _service: serv, firstname: first,
lastname: last}, function(result){
        $("span").html(result);
     });
  });
});
</script>
</head>
<body>
```

```
Enter Lastname:
<input type="text" id="last">
Enter Firstname:
<input type="text" id="first">

<button>Insert Data</button>
<span></span>
<div></div>
</body>
</html>
```

This HTML file creates a post request to the server which will execute the program test2.sas which contains:

```
data names;
length firstname lastname $ 50;
run;
proc sql undo_policy=none;
insert into work.names
set
lastname="&lastname",
firstname="&firstname";
quit;
proc json out=_webout;
export names / nokeys nosastags;
run;
```

The program test2.sas first creates an empty data set with 2 variables. The next statement (proc sql) inserts the contents of the macro variables lastname and firstname into a new row in the data set. The final statement is just a way to show that the variables were successfully saved into the data set by exporting them using the JSON procedure.

# **CONCLUSION**

SAS/IntrNet is still one of my favorite tools for developing and deploying web applications and web content. Adding the ever-changing JavaScript libraries, like jQuery AJAX, and using tools such as PROC JSON, developing web sites or web services is a really quite easy. Most of the development of these pages took only a few minutes to complete and even less time to deploy.

#### **REFERENCES**

W3C Schools . "jQuery Tutorial" 1996-2015 Accessed September 2015 http://www.w3schools.com/jquery/.

Oracle Java EE 6 Tutorial 2013 "What are RESTful Web Services" Accessed September 2015 http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html

JSON.org "Introducing JSON" Accessed September 2015 http://www.json.org/

# **RECOMMENDED READING**

Base SAS<sup>®</sup> 9.4 Procedures Guide JSON Procedure

## **CONTACT INFORMATION**

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

Jeremy Palbicki Mayo Clinic (507)538-1428 palbicki.jeremy@mayo.edu

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.