Paper RI-07

Enhance Your SAS/Intrnet® Application with jQuery and Google Earth

David Mintz, U.S. Environmental Protection Agency, Research Triangle Park, NC

ABSTRACT

Sometimes the hardest part of developing a data delivery system is designing the interface. How will users select what they need? How will the output be displayed? What's the most efficient design when menu options need to reflect database updates? This paper will demonstrate how to enhance your SAS/Intrnet application with jQuery and a Google Earth API to satisfy common user expectations. Examples will show how to implement data driven menu options that change as your data change, return output to the same window (with no page refresh!), display a "processing data" icon while your program is running, and provide access to data-driven KML files via a map API.

INTRODUCTION

The EPA recently updated a SAS/Intrnet application which serves as a primary data delivery and visualization system for air quality data in the United States. I was lucky enough to serve as a SAS programmer/developer on the team and would like to share some of the solutions we implemented in hopes they might benefit the SAS Web developer community. If you want to see some functional examples before deciding whether to continue reading this paper, you can view the application at http://www.epa.gov/airdata.

We started with some key functional requirements. First, we needed results to be displayed in the same window with no page refresh. Second, we needed data-driven, dependent select lists. Third, we needed a "processing data" icon to display while users await results. Fourth, we needed a way to display and overlay existing KML files within a web browser. Finally, whatever technology we selected needed to be compatible across all major Web browsers.

After exploring several ways to implement the first three requirements, it became clear that jQuery was the perfect solution for three main reasons. First, the code is relatively simple and clean, making it easy to implement and maintain. Second, it's compatible across all major browsers, which is probably why it's used by so many popular enterprises like Amazon, Netflix, IBM, Dell, Bank of America, and Best Buy just to name a few. Finally, it enables web page content to be updated without a page refresh.

So, what is jQuery? Simply put, jQuery is a JavaScript framework. Essentially, it executes JavaScript code (in most cases several lines of complex JavaScript code) with a simple jQuery script. To us SAS programmers, it's akin to calling a macro. This paper is certainly not meant to teach jQuery or even be a tutorial. (Admittedly, I'm a jQuery amateur, bordering on novice.) It simply touches on a few key components – enough to show how it can work effectively with SAS/Intrnet applications. If you really want to dive into jQuery, I highly recommend *jQuery in Action* which is referenced at the end of this paper.

So, what about the solution for displaying KML files, our fourth functional requirement? Previously, we had posted KML files on a web page for users to open with Google Earth or other KML viewers. However, we wanted folks to be able to use the maps inside the web browser without having to open another application. When the Google Earth API was released, it seemed to be a good solution. The developer's page <https://developers.google.com/earth/> has some very good examples which we were able to customize. One final introductory note – this paper assumes that you have an existing SAS/Intrnet application or that you are familiar with how they work.

DATA-DRIVEN SELECT LIST USING JQUERY AND SAS

If you're familiar with how the SAS/Intrnet Application Dispatcher works, then you know you can invoke a SAS/Intrnet program with a URL string that includes the service name and any name-value pairs. The string can be passed to SAS via an html form or any Web browser's address bar. You can also embed the string inside an html <a> or <img> tag to render a dynamic link or image. Likewise, you can use jQuery to invoke a SAS/Intrnet program and even build the string that passes name-value pairs. Let's look at how jQuery can be used with SAS/Intrnet to populate a select list dynamically.

Step1. Name the elements

For simplicity, let's name three select tags – pollutant, year, and county. In a static html form, <option> tags would follow each <select> tag. However, the <option> tags will be added dynamically. So, just add the </select> close tag.

```
…
```

```
<body>
…
<select id="pollutant"></select>
<select id="year"></select>
<select id="county"></select>
…
</body>
…
```

Step2. Add jQuery code

The two jQuery methods I use for invoking SAS/Intrnet programs are load and serialize. They execute Ajax (Asynchronous JavaScript and XML) events, which means they are able to pass requests to the server and then load responses back in the page. The load method simply grabs content and stuffs it into the appropriate element. I use the load method when I need to pass a specific URL with fixed name-value pairs. I use the serialize method, in conjunction with the load method, when I need to collect and pass all valid name-value pairs the user has selected. The serialize method automatically creates a formatted and encoded query string for you. The best part is that the results are returned to the element with no page refresh. Here's the code. Note that there are a couple of additional lines of code, not included here, that are needed to point to the jQuery library.

```
<script type="text/javascript">
    $(function() {
          $('#poll').load('http://yourserver/cgi-bin/broker?_service=yourservice
&_program=dataprog.load_pollutants.sas');
          $('#poll').change(function(event){
                    $('#year').load('http://yourserver/cgi-
bin/broker?_service=yourservice&_program=dataprog.load_years.sas',
                         $(this).serialize()
                         );
                               });
          $('#year').change(function(event){
                    $('#county').load('http://yourserver/cgi-
bin/broker?_service=yourservice&_program=dataprog.load_counties.sas',
                         $('form').serialize()
                         );
                               });
          });
</script>
```

Step 3. Create SAS programs that make the <option> tags

Now you just need SAS programs that accept the selections and build the option tags. Here's the code for our example. Each one creates a list of <option> tags that are loaded into the page by the jQuery load method based on user selections.

```
*from load_pollutants.sas;
%macro loadpollutants;
data _null_;
     length options $100.;
     file _webout;
     keep options;
     set lib.pollutants end=last;
     if _n_=1 then do;
          options='<option value="-1">Select ...</option>';
          put options;
     end;
     options='<option
value="'||trim(left(poll))||'">'||trim(left(pollutant))||'</option>';
     put options;
run;
%mend loadpollutants;
%loadpollutants run;
```

```
*from load_years.sas;
%macro loadyears;
data getyears;
     keep year;
     if poll='PM10' then styear=1988;
     else if poll='PM2.5' then styear=1999;
     else styear=1980;
     thisyear=year(date());
     do year=styear to thisyear;
          output;
     end;
run;
proc sort data=getyears;
     by descending year;
run;
data _null_;
     length options $50.;
     file _webout;
     keep options;
     set getyears;
     if _n_=1 then do;
          options='<option value="-1">Select ...</option>';
          put options;
     end;
     options='<option>'||put(year,$4.)||'</option>';
     put options;
run;
%mend loadyears;
%loadyears run;


*from load_counties.sas;
%macro loadcounties;
proc sql;
     create table counties as
          select distinct put(a.countyCode,z5.) as countycode, b.countyName
        from lib.cbsa_site_poll a, lib.counties b
        where a.parm in (%unquote(%str(%'&poll%'))) and a.countyCode=b.countyCode
                  and a.y&year is not null;
quit;
%end;
data _null_;
     length options $150.;
     file _webout;
     keep options;
     set counties;
     if countycode ne .;
     if _n_=1 then do;
          options='<option value="-1">Select a County ...</option>';
          put options;
     end;
     options='<option
value="'||put(countycode,$5.)||'">'||trim(left(countyname))||'</option>';
     put options;
run;
%mend loadcounties;
%loadcounties run;
```

Now that everything is in place, let's look at it from the user perspective and review what happens. When the html page loads in a browser, the load method for the "pollutant" element calls the "load_pollutants.sas" program. The output from that program is a list of <option> tags which is passed back to the html page and displayed in the "pollutant" select tag. Viola – a dynamic select list. But that's just the beginning! The first change method in the

jQuery code says when a user selects a pollutant (or changes that selection), then use the serialize method to collect that name-value pair and pass it as an argument in the load method for "year". The load method takes that information and runs "load_years.sas". That program accepts the pollutant value selected and returns the list of years valid for that pollutant. A dependent, dynamic select list! Finally, the second change method says when the "year" selection changes, then use the serialize method again to collect all valid name-value pairs (now there are two pairs – one for pollutant and one for year) and pass them as arguments in the load method for "county". The load method takes that information and runs "load_counties.sas". That program references a SAS data set and returns only counties having data for the pollutant and year selected. And, did I mention all this happens with no page refresh? Fantastic!

Of course, this is a simplified example. You can see an actual production page and view the source code at http://www.epa.gov/airdata/ad_viz_tile.html or similar pages on that site.

RETURNING SAS/INTRNET RESULTS TO THE SAME WINDOW

Since the jQuery load method can be used to grab content and pull it back into the page, it's a great way to bring SAS/Intrnet results into the same window (say it with me) with no page refresh. Like the previous example, you just need a container html tag and jQuery code that says where to put it. Let's look at an example.

Step 1. Name the elements

Here, we'll actually name two elements. The first, called "launch", is a button for the user to click. The second, called "results", is a place holder for the SAS/Intrnet results. These can be inserted into the html code directly beneath the <select> tags in the previous example.

```
<div id="launch"><input type="button" value="Plot Data" /></div>
<div id="results"></div>
```

Step 2. Add the jQuery code

Here's the code which can be inserted in the jQuery script in the previous example. Notice, we use the load and serialize methods again. When the "Plot Data" button is clicked, the load method runs "ad_viz_tile.sas" using all valid name-value pairs which are automatically collected from the form by the serialize method. The results from the program are loaded into the "results" element – you guessed it – with no page refresh.

```
$('#launch').click(function(event){
    $('#results').load('http://yourserver/cgi-
bin/broker?_service=yourservice&_debug=0&_program=dataprog.ad_viz_tile.sas',
           $('form').serialize()
                                );
    $(this).hide();
                         });
```

DISPLAYING A "PROCESSING DATA" ICON

How do users know their request is being processed? With one additional line of jQuery, your application can display a "processing data" message to let them know. The new line is in bold below. It simply displays content in the "results" element when the button is clicked. The content is up to you. In this example, a "Retrieving Data" text message displays along with an animated gif file commonly used in many popular apps. As soon as the SAS program finishes, the results replace those items.

```
$('#launch').click(function(event){
    $('#results').html('<p>Retrieving Data...</p><img
src="http://yourserver/images/ajax-loader.gif" />');
    $('#results').load('http://yourserver/cgi-
bin/broker?_service=yourservice&_debug=0&_program=dataprog.ad_viz_tile.sas',
           $('form').serialize()
                                );
    $(this).hide();
                         });
```

DISPLAYING KML FILES WITH A MAP API

As you probably know, there are many ways to provide a map interface. We chose this method primarily because it links to existing KML files and allows each map layer to be toggled on and off. You can access the working example at http://www.epa.gov/airdata/ad_maps.html. Let's take a look at what's involved. You need to know a little bit of html and JavaScript, but that's all. It does not employ jQuery, but I suppose it could.

Step 1. Make sure you have KML files

Our KML files are generated by a SAS program that runs as a weekly cron job. The program queries a database to pull in any new point information since the previous run. Each week, the new KML files overwrite the previous files and another cron job actually zips them into KMZ files for more efficient use in the API. For details on how to generate a KML file using SAS (and even how to connect your KML files to your SAS/Intrnet service), see the 2008 SAS paper listed in the References section.

Step 2. Add the JavaScript code

The following code points to the KML files and allows them to be toggled on and off. We also added a few other features like the highway overlay and the search box allowing users to zoom to a particular landmark. You'll find these and more examples at https://developers.google.com/earth/.

```
<body class="basic wide" onLoad="init();">
    ...

    <div style="clear: both;"></div>
    <div id="ui" style="position: relative;">
          <div id="map3d_container" style="border: 1px solid #000; width: 730px;
height: 500px;">
          <div id="map3d" style="height: 100%;"></div>
      </div>

    <div id="extra-ui" style="position: absolute; left: 750px; top: 0; width:
450px;">
      <input id="location" type="text" value="" alt="search this map" size="15" />
      <input type="submit" onClick="buttonClick()" value="Go"/><br/>
      <span class="fileinfo">Zoom to a city, county, address, zip code, or
lat/lon</span><br/>
      <br/>
      Monitoring Networks<br />

  <input type="checkbox" id="kml-Ozone-check" onClick="toggleKml('Ozone');"/>
  <label for="kml-Ozone-check"><a
href="http://yourserver/Ozone.kmz">Ozone</a></label>
  <img src="http://yourserver/mapicons/pink.png" width="15" height="13" /><span
class="fileinfo">active</span>
  <img src="http://yourserver/mapicons/pink-dot.png" width="15" height="13" /><span
class="fileinfo">inactive</span><br/>

  <input type="checkbox" id="kml-SO2-check" onClick="toggleKml('SO2');"/>
  <label for="kml-SO2-check"><a href="http://yourserver/SO2.kmz">SO2</a></label>
  <img src="http://yourserver/mapicons/purple.png" width="15" height="13" /><span
class="fileinfo">active</span>
  <img src="http://yourserver/mapicons/purple-dot.png" width="15" height="13"
/><span class="fileinfo">inactive</span><br/>

     </div>
     </div>

    <!-- google earth scripts -->
  <style type="text/css">
    @import "static/examples.css";
    </style>
  <style type="text/css">
```

```
  @import "static/prettify/prettify.css";
  </style>
<script type="text/javascript" src="static/prettify/prettify.js"></script>

<script type="text/javascript">
/* <![CDATA[ */
var ge;

// store the object loaded for the given file... initially none of the objects
// are loaded, so initialize these to null
var currentKmlObjects = {
  'Ozone': null,
  'SO2': null
};

google.load("earth", "1");
google.load("maps", "2.160");

function init() {
  google.earth.createInstance('map3d', initCB, failureCB);
}

function initCB(instance) {
  ge = instance;
  ge.getWindow().setVisibility(true);

  // add a navigation control
  ge.getNavigationControl().setVisibility(ge.VISIBILITY_AUTO);
  //ge.getNavigationControl().setControlType(ge.NAVIGATION_CONTROL_SMALL);

  // add some layers
  ge.getLayerRoot().enableLayerById(ge.LAYER_BORDERS, true);
  ge.getLayerRoot().enableLayerById(ge.LAYER_ROADS, true);

  // fly to US
  var la = ge.createLookAt('');
  la.set(38, -98,
    0, // altitude
    ge.ALTITUDE_RELATIVE_TO_GROUND,
    0, // heading
    0, // straight-down tilt
    3500000 // range (inverse of zoom)
    );
  ge.getView().setAbstractView(la);

  // if the page loaded with checkboxes checked, load the appropriate
  // KML files

  if (document.getElementById('kml-Ozone-check').checked)
    loadKml('Ozone');

  if (document.getElementById('kml-SO2-check').checked)
    loadKml('SO2');

  document.getElementById('installed-plugin-version').innerHTML =
    ge.getPluginVersion().toString();
}

function failureCB(errorCode) {
}

function toggleKml(file) {
  // remove the old KML object if it exists
```

```
  if (currentKmlObjects[file]) {
    ge.getFeatures().removeChild(currentKmlObjects[file]);
    currentKmlObject = null;
  }

  // if the checkbox is checked, fetch the KML and show it on Earth
  var kmlCheckbox = document.getElementById('kml-' + file + '-check');
  if (kmlCheckbox.checked)
    loadKml(file);
}

function loadKml(file) {
  var kmlUrl = 'http://yourserver/' + file + '.kmz';

  // fetch the KML
  google.earth.fetchKml(ge, kmlUrl, function(kmlObject) {
    // NOTE: we still have access to the 'file' variable (via JS closures)

    if (kmlObject) {
      // show it on Earth
      currentKmlObjects[file] = kmlObject;
      ge.getFeatures().appendChild(kmlObject);
    } else {
      // bad KML
      currentKmlObjects[file] = null;

      // wrap alerts in API callbacks and event handlers
      // in a setTimeout to prevent deadlock in some browsers
      setTimeout(function() {
        alert('Bad or null KML.');
      }, 0);

      // uncheck the box
      document.getElementById('kml-' + file + '-check').checked = '';
    }
  });
}

function buttonClick() {
  var geocodeLocation = document.getElementById('location').value;

  var geocoder = new google.maps.ClientGeocoder();
  geocoder.getLatLng(geocodeLocation, function(point) {
    if (point) {
      var lookAt = ge.createLookAt('');
      lookAt.set(point.y, point.x, 0, ge.ALTITUDE_RELATIVE_TO_GROUND,
                 0, 0, 5000);
      ge.getView().setAbstractView(lookAt);
    }
  });
}

/* ]]> */
  </script>
  <!-- end google earth scripts -->
```

CONCLUSION

User interfaces for SAS/Intrnet applications can be enhanced by thoughtfully employing open source technology in combination with SAS software. Technology changes so rapidly that I wonder how long this paper and others like it will be relevant. Time will tell. In the meantime, it is my hope this paper will serve as a useful reference for enhancing your SAS/Intrnet applications.

REFERENCES

- Bibeault, B. & Katz, Y. (2010). *jQuery in Action*, Second Edition. Stamfort, CT: Manning Publication Co.
- Drukenbrod, J. & Mintz, D. (2008), Using SAS® and Google Earth to Access and Display Air Pollution Data, SAS Global Forum 2008 Proceedings.
- Google Earth API Examples. https://developers.google.com/earth/documentation/examples.

ACKNOWLEDGMENTS

I would like to acknowledge the EPA team who redesigned the AirData website - Kristen Bremer, Halil Cakir, Rick Copland, Jan Cortelyou, Josh Drukenbrod, Tim Hanley, Nick Mangus, Jonathan Miller, Tesh Rao, Michelle Wayland, and Rob Wildermann.

CONTACT INFORMATION

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

Name: David Mintz
Enterprise: U.S. Environmental Protection Agency
Address: 109 TW Alexander Drive
City, State ZIP: Research Triangle Park, NC 27711
Work Phone: 919-541-5224
E-mail: mintz.david@epa.gov