Paper 015-2009

Geospatial Analysis Using SAS[®] and the Google Map API Mai Nguyen, Shane Trahan, Patricia Nguyen, Wafa Handley RTI International, RTP, NC

ABSTRACT

The Google Map API allows Web site developers to integrate mapping capabilities into their own Web sites. These mash-ups combine data from more than one source into a single tool used by a broad range of people including real estate professionals who map properties for potential clients to municipalities which relay traffic-related issues through their Web sites. RTI International researchers have taken this a step further by integrating the Google API into a library that can be used not only by Web developers but by SAS programmers and other developers as well. SAS programmers can take advantage of the Google API to perform a variety of Geographic Information System (GIS) calculations by using a set of easy to use function calls in the data step.

Our paper will provide two SAS programs showing (1) how to obtain Geocode information for specific addresses and (2) how to compute the crow-fly and the traveling distance between any two locations. We will briefly discuss the technical approach used allowing SAS programs to access these Web-based APIs.

INTRODUCTION

The Google Map Application Programming Interface (API) is a free service created by Google providing developers with the capability to integrate Google Maps into their own websites. The maps are used in a myriad of applications ranging from advanced real estate search applications to simply finding the location of your local Starbucks coffee shop. Google Maps uses JavaScript extensively as the scripting language for client-side web based development. Acquiring access to the maps is fairly uncomplicated, but does require a developer to obtain a free access key from Google. Once this key has been obtained, using the Google Map API requires a simple HTML script tag as shown below:

```
<script type="text/javascript"
src=http://www.google.com/jsapi?key=ABCDEF></script>
```

With the loaded API, the client browser can access all of the functionality for Google Maps.

The Google Maps API allows developers to perform a variety of GIS calculations utilizing data points found in the map images. Our paper will focus on two major concepts utilizing these capabilities, 1) geocode, which is the process of converting geographic information into latitude and longitude coordinates and 2) using this geocode information to acquire traveling distance between two points. Our SAS programs will demonstrate the use of the SAS[®] JavaObj as an interface for SAS programs to obtain geocode information and calculate distances between these two locations.

INTEGRATING GOOGLE MAP API AND SAS

RTI has created the RTI Google Map Library using the Microsoft .NET framework enabling programs to use the Google Map API as a service. Programs using the RTI Google Map Library can be written in .NET, Java or SAS. The structure of the library is shown below in Figure 1



Figure 1 - RTI Google Map Library

The RTI Google Map Library consists of several layers built on top of the Microsoft .NET libraries. The Javascript Handler layer is responsible for all communication with Google servers using the Google Map API. The .NET library provides the abstraction and class library for .NET GIS applications. The Java Interface consists of proxy classes bridging Java and SAS applications with the .NET Library.

The SAS JavaObj object provides a programmatic interface for SAS programs to interact with other systems or programming libraries written in Java. Using the JavaObj, SAS programs can easily instantiate a Java object and access its variables and methods. There have been a number of papers and resources available from SAS and related conferences providing great detail about how to use JavaObj to extend capabilities of the SAS platform. Included below are some simple SAS code snippets as an introduction to the JavaObj for those unfamiliar with this SAS object.

The code snippet below declares a JavaObj object and then instantiates a Java String object with the value of 'ABC':

```
declare javaobj j;
j = _NEW_ javaobj('java/lang/String', 'ABC');
```

The same can be done in a more compact way:

```
declare JavaObj j('java/lang/String', 'ABC');
```

We can easily call the "concat" method on the Java String object and put the concatenated string in a SAS variable:

```
Length s_out $20;
j.callStringMethod('concat', '123', s_out);
```

The value of s_out is equal to 'ABC123'.

The Delete method should be used to dispose the JavaObj once the task is done:

j.Delete();

JAVA CONFIGURATION WITH SAS

Although the JavaObj is still experimental the support for Java type API calls is starting to become more widely adopted and the code required to access Java objects within the SAS data step is easy and straight forward. The new component interface was added as a new feature in version 9. Our code examples and the work reflected in this paper use SAS 9.2. It is recommended that the latest release of the Java Runtime Environment (JRE) is used which,

as of the writing of this paper, is version 6. The JRE is freely available and can be downloaded from http://www.java.com.

In order for SAS to correctly work with Java class libraries the SAS environment must be made aware of the location of Java classes and/or Java Archive or JAR files to be used at runtime. This can be done by updating the JREOPTIOPNS variable in the SASV9.CFG file located in the SAS root or in the corresponding language folder (c:\program files\SAS\SAS 9.1\nls\en). In many instances the initial SAS installation will have the basic JREOPTIONS settings in the configuration file but for reference it is included here.

```
/* Options used when SAS is accessing a JVM for JNI processing */
-JREOPTIONS=(
-Dsas.jre.home=C:\PROGRA~1\SAS\SHARED~1\JRE\1499C1~1.2_0
-Djava.security.policy=!SASROOT\core\sasmisc\sas.policy
-Dsas.app.class.dirs=!SASROOT\core\sasmisc;
C:\PROGRA~1\SAS\SHARED~1\applets\9.1
-Dsas.jre=private -Dsas.ext.config=!SASROOT\core\sasmisc\sas.java.ext.config
-DPFS_TEMPLATE=!SASROOT\core\sasmisc\qrpfstpt.xml
-Djava.class.path=!SASROOT\core\sasmisc\sas.launcher.jar
-Djava.system.class.loader=com.sas.app.AppClassLoader
)
```

The above syntax would all be on one line in the SASV9.CFG file but it is on multiple lines here for easier reading. To add in your own class or JAR files a new path needs to be added to the JREOPTIONS environment control by adding in a new option inside the parenthesis of the JREOPTIONS.

```
-Dsas.app.class.path=[ClassPath value]
```

For example, if the class files are located at c:\projects\SASGIS then the new JREOPTIONS operation would include the following

```
-Dsas.app.class.path=C:\Projects\SasGis
```

RTI GOOGLE MAP LIBRARY JAVA INTERFACE

The Java Interface layer of the RTI Google Map library provides a simple interface allowing SAS programs to easily use the Google Map API. Currently the Java API consists of 3 main Java classes: GeoCode, TravelDistance and CrowFlyDistance. The complete class API reference is included at the end of this section.

As shown below, the GeoCode class is very simple. The actual complicated work is performed within the .NET library

```
public class GeoCode {
    private String address ;
    private String status ;
    private double latitude ;
    private double longitude ;

    public GeoCode() {
        address = null ;
        status = "-1" ;
        latitude = 0.0 ;
        longitude = 0.0 ;
    }

    public GeoCode(String streetAddress) {
        this.GetGeoCode(streetAddress) ;
    }

    public String GetGeoCode(String streetAddress)
    {
}
```

```
address = streetAddress ;
       this.GetGeoCode() ;
       return this.status ;
}
private void GetGeoCode()
       // Get lat and long coordinate
       String progName = "C:\\Projects\\SasGis\\GeoCode4SAS.exe " ;
       String param = "\"" + address + "\"";
       String coordinate = CmdExec.Run(progName + param);
       // parse result
       String[] s = coordinate.split(",") ;
       status = s[0];
       if (status.compareTo("200") == 0)
       {
             latitude = Double.parseDouble(s[1]) ;
             longitude = Double.parseDouble(s[2]) ;
       }
}
public double GetLat() {
       return latitude ;
}
public double GetLong() {
      return longitude ;
}
public String GetAddress() {
       return address ;
}
public String GetStatus() {
      return status ;
}
public String GetCoordinate() {
       return String.valueOf(latitude) + "," + String.valueOf(longitude);
}
```

The GeoCode class provides 2 constructors; the first one takes no parameters and is used to initialize the instance of the Geocode object. The other constructor accepts an address as a parameter. This constructor will call the GetGeoCode method to obtain the geocode immediately.

The GeoCode class provides a public GetGeoCode method to obtain geocodes for specified addresses passed in as arguments. It also includes a small number of getter methods for getting status codes and coordinates.

This simple pattern is also used for the TravelDistance and CorwFlyDistance classes.

RTI GOOGLE MAP LIBRARY API REFERENCE

Listed below are the API references for the GeoCode, CrowFlyDistance, and TravelDistance classes.

Class GeoCode

}

The GeoCode class provides the geocode for a specified address.

Constructor

Constructor	Description
GeoCode()	Initializes an instance of the GeoCode object
GeoCode(string streetAddress)	Instantiates an instance of the GeoCode object

and obtains the geocode for the streetAddress
parameter

Public Methods

Method	Return Value	Description
GetGeoCode(string streetAddress)	String	Obtains the geocode for the streetAddress. It returns a status code indicating if the geocode search was successful. Refer to GetStatus() method for status codes.
GetLat()	Double	Returns the latitude coordinate
GetLong()	Double	Returns the longitude coordinate
GetAddress()	String	Returns the address for which a GeoCode is being searched
GetStatus()	String	Returns the search status of an address. 200 indicate an address was successfully parsed and a geocode value returned. Status codes include • 200 Success • 400 Bad Request • 500 Server Error • 601 Missing Address • 602 Unknown Address • 603 Unavailable Address • 604 Unknown Directions • 610 Bad Key • 620 Too Many Queries
GetCoordinate()	String	Returns the coordinate value. Combined latitude and longitude values.

Class CrowFlyDistance

The CrowFlyDistance class will return the distance in double format in unit miles given two valid parsable addresses. The CrowFlyDistance is a straight line reference from the GetFromAddres to the GetToAddress address values.

Constructor

Constructor	Description
CrowFlyDistance()	Initializes an instance of the CrowFlyDistance object
CrowFlyDistance(string	Instantiates an instance of the CrowFlyDistance object
fromAddr, string toAddr)	and obtains the crowflydistance between the fromaddress
	and toaddress parameters

Public Methods

Method	Return Value	Description
GetDistance(string fromAddr, string toAddr)	String	Obtains the CrowFlyDistance between two addresses. This will return a status code indicating if a distance

		value was found. Refer to GeoCode Class GetStatus() method for status codes.
GetMiles ()	Double	Returns the distance in miles.
GetFromAddress()	String	Returns the from address value being used to calculate the distance.
GetToAddress()	String	Returns the to address value being used to calculate distance.
GetStatus()	String	Returns the result of the distance calculation. A value of 200 indicates address information was successfully parsed and a distance value returned. Refer to GeoCode Class GetStatus() method for status codes.

Class TravelDistance

The TravelDistance class will return the distance in double format in unit miles given two valid parsable addresses. The TravelDistance is the distance of travel required from the GetFromAddres to the GetToAddress address values via known roads to the Google Maps API.

Constructor

Constructor	Description
TravelDistance ()	Initializes an instance of the TravelDistance object
TravelDistance(string	Instantiates an instance of the TravelDistance object
fromAddr, string toAddr)	and obtains the travel distance between the
	fromaddress and toaddress parameters

Public Methods

Method	Return Value	Description
GetDistance(string fromAddr, string toAddr)	String	Obtains the travel distance between two addresses. This will return a status code indicating if a distance value was found. Refer to GeoCode Class GetStatus() method for status codes.
GetMiles()	Double	Returns the distance in miles
GetFromAddress()	String	Returns the from address being used to calculate the distance

GetToAddress()	String	Returns the to address being used to
GetStatus()	String	Returns the result of the distance calculation. A value of 200 indicates address information was successfully parsed and a distance value returned. Refer to GeoCode Class GetStatus() method for status codes.

EXAMPLES

Our first example is a SAS program that obtains the geocode information (longitude and latitude coordinate). Our second example will demonstrate distance calculations between 2 locations.

EXAMPLE 1 – OBTAINING GEOCODE INFORMATION

In this example, we demonstrate how to use the RTI Google Map Library to obtain geocode data for an address. This is a simple 3-step process:

- 1. Create a JavaObj and instantiate the GeoCode Java object from the RTI Google Map Library. The address is passed to the Java object constructor as a string parameter.
- 2. Call the GetStatus method on the Java GeoCode object to get the status of the operation
- 3. If the status indicates that a geocode was successfully returned, call the GetLat or GetLong method to get the latitude or longitude value respectively. Alternatively, we can also call the GetCoordinate method to get both the latitude and longitude in the coordinate format.

```
libname out 'c:\projects\SasGis';
```

```
data temp;
   input street1 $ 1-20 city1 $ 21-30 state1 $ 31-33 zip1 $ 34-38
         street2 $ 41-70 city2 $ 71-80 state2 $ 81-84 zip2 $ 84-88 ;
   datalines;
303 gingergate dr cary
                         nc 27519 11 W. Jones Street
                                                                raleigh nc 27601
108 ashley glen dr cary
                         nc 27513 8611 Brier Creek Parkway
                                                                raleigh nc 27617;
run ;
data out.Geocode;
   set temp;
   length status1 $3 lat1 8 long1 8 coordinate1 $25
          status2 $3 lat2 8 long2 8 coordinate2 $25;
   /* Declare java object */
   declare javaobj j1;
   /* Create GeoCode java object */
   j1 = _NEW_ JavaObj('GeoCode', street1||city1||state1||zip1);
   /* Get status */
   j1.callStringMethod('GetStatus', status1);
   if (status1 = '200') then
   do ;
          /* Found geocode */
          j1.callDoubleMethod('GetLat', lat1);
          j1.callDoubleMethod('GetLong', long1);
          j1.callStringMethod('GetCoordinate', coordinate1);
```

```
end;
   /* Geocode the second address */
   j1.callStringMethod('GetGeoCode',
           street2||city2||state2||zip2, status2);
   if (status2 = '200') then
   do ;
          /* Found geocode */
          j1.callDoubleMethod('GetLat', lat2);
          j1.callDoubleMethod('GetLong', long2) ;
          j1.callStringMethod('GetCoordinate', coordinate2);
   end;
   j1.Delete();
run ;
proc print data=out.GeoCode;
   Title 'Dataset OUT.GEOCODE';
run;
```

Note that once the GeoCode object has been instantiated, its API includes the GetGeoCode method that can be used for geocoding new address without having to instantiate a new GeoCode object. This is used to obtain the geocode for the second address in our example.

EXAMPLE 2 – GETTING THE DISTANCES

In this section, we demonstrate how to obtain either the crow-fly or the traveling distance between 2 locations by using the RTI Google Map Library.

The RTI Google Map API provides a similar pattern for calculating the distance between 2 locations:

- 1. Create a JavaObj and instantiate the TravelDistance or CrowFlyDistance Java object from the RTI Google Map Library. The "from address" and "to address" are passed to the Java object constructor as parameters.
- 2. Call the GetStatus method on the Java object to get the status of the operation.
- If the status indicates that the distance was successfully calculated, call the GetMiles method to get the distance value. Both TravelDistance and CrowFlyDistance objects expose the exact same interface, but internal implementations of each object are specific to its task.

```
data out.Distance(drop = address1 address2);
      set out.GeoCode;
      length DistStat $3 TravelDist 8 CrowFlyDist 8;
      if (Status1 = '200' and Status2 = '200') then
      do;
             declare javaobj j1;
             address1 = street1
                                 || city1
                                             state1
                                                       zip1;
             address2 = street2 || city2
                                          || state2
                                                       zip2;
              /* Get traveling distance */
             j1 = _NEW_ JavaObj('TravelDistance', address1,
                                    address2);
             j1.callStringMethod('GetStatus', DistStat) ;
             if (DistStat = '200') then
             do ;
                     /* Get travel distance */
                    jl.callDoubleMethod('GetMiles', TravelDist) ;
             end;
             j1.Delete();
             /* Get crow-fly distance */
             j1 = _NEW_ JavaObj('CrowFlyDistance', address1,
                                   address2);
             j1.callDoubleMethod('GetMiles', CrowFlyDist) ;
```

j1.Delete();

end;

run;

EXAMPLE OUTPUT

Figure 2 below displays the final dataset of the SAS output from a PROC PRINT. The output shown here is divided into three parts. The first top most part of the output is the address information that was used. The second set of information is the latitude and longitudinal coordinates that were used to calculate distances. The third and final set of information shown lists coordinates used and results of the crow-fly and travel distance calculations.

Crow-fly calculations are defined as the shortest distance between any two points where the travel distance is considered to be the travel distance using known roads including interstates highways and other surface transportation.

0bs	street1		city1	state1	z ip1	stree	t2		city2
1 2	303 ginge 108 ashle	ergate dr ey glen dr	cary cary	nc nc	27519 27513	11 W. 8611	Jones Stre Brier Creek	et Parkway	raleigh raleigh
Obs	state2	z ip2	status1	lat1	long1		coordinat	e1	status2
1 2	nc 2 nc 2	27601 27617	200 200	35.7867 35.8071	-78.8532 -78.7991	35 35	.786718,-78 .80705478	.853163 .799051	200 200
Obs	lat2	long2		coordinate2		Dist Stat	Travel Dist	Crow FlyDist	
1	35.7826	-78.639	1 35.7	825554, - 78.63	90509	200	18.3391	12.0050	

Figure 2 – Example of Output from PROC PRINT

It should be remembered that although output in figure 2 is shown from a PROC PRINT, this information can be used in a number of ways including incorporating results into databases, spreadsheets or any other type of application. The RTI Google Map Library is meant to be a feature rich robust library used across a myriad of languages and applications.

CONCLUSION

Our examples demonstrate how the Java API from our RTI Google Map Library is used to perform geospatial calculations in a SAS program. The RTI Google Map API is intuitive and easy-to-use. All programming complexities and network handling are transparent to the SAS programmers, allowing a variety of users the ability to exploit the powerful GIS capabilities of the Google Map API.

Integrating SAS capabilities with java and Google Map API opens up new venues for SAS users, in fields such as epidemiology, demography and market research among other types of geospatial analysis fields. The creation of datapoints and waypoints from a variety of devices could be combined with address specific data to assist businesses and policy makers in decision making. The majority of GIS mapping systems have the capability to import and export mapping data such as GeoCode information. The tools that have been highlighted here allow for closer collaboration between GIS analysts and SAS developers.

True applications of Geospatial analysis are still in their infancy. Yet, with increased geographical data becoming available, there is a crucial need in both the public and private sectors to make efficient use of this information. Incorporating the statistical power of SAS to this type of data will no doubt open many more doors in a variety of research areas many of which currently use limited geospatial data. Our aim is to allow SAS developers to further enrich research by broadening data availability. This paper highlights a data acquisition model that is inclusive of SAS, .NET or Java developers. As more data becomes available from web based services, this type of model will be beneficial to these development communities.

RECOMMENDED READING

Google Inc. 2008. "Google Maps API." http://code.google.com/apis/maps/.

SAS Institute Inc. "SAS 9.2 Companion for Windows" http://support.sas.com/documentation/cdl/en/hostwin/59544/HTML/default/win-sysop-jreoptions.htm

SAS Institute Inc "SAS 9.2 Language Reference: Dictionary" <u>http://support.sas.com/documentation/cdl/en/Irdict/59540/HTML/default/a002587970.htm</u>

DeVenezia, Richard A, "Java in SAS JavaObj, A DATA Step Component Object" <u>http://www2.sas.com/proceedings/sugi30/241-30.pdf</u>

CONTACT INFORMATION

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

Name: Mai Nguyen	Name: Shane Trahan
Enterprise: RTI International	Enterprise: RTI International
Address: 3040 Cornwallis Rd	Address: 3040 Cornwallis Rd
City, State ZIP: Durham NC 27709	City, State ZIP: Durham NC 27709
Work Phone: 919-541-8757	Work Phone: 919-541-5848
Fax: 919-541-6178	Fax: 919-541-6178
E-mail: mnguyen@rti.org	E-mail: srt@rti.org
Web: http://www.rti.org	Web: http://www.rti.org
Name: Patricia Nguyen	Name: Wafa Handley
Enterprise: RTI International	Enterprise: RTI International
Address: 3040 Cornwallis Rd	Address: 3040 Cornwallis Rd
City, State ZIP: Durham NC 27709	City, State ZIP: Durham NC 27709
Work Phone: 919-541-5713	Work Phone: 919-541-6066
Fax: 919-541-6178	Fax: 919-541-6178
E-mail: pnguyen@rti.org	E-mail: handley@rti.org
Web: http://www.rti.org	Web: http://www.rti.org

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.