Paper 253-2008

Using SAS® and Google Earth to Access and Display Air Pollution Data

Josh Drukenbrod and David Mintz, U.S. Environmental Protection Agency
Office of Air Quality Planning Standards, Research Triangle Park, NC

ABSTRACT

If you have ever spent large amounts of time decrypting dense, hard-to-read tables of data, then visualization tools are more appealing than ever. As the demand to visualize data increases, the market for these tools will continue to expand. Recognizing potential applications for studying air data, EPA's Emissions Inventory and Analysis Group recently built a visualization tool which exploits a previously unexploited mutual relationship between one of the world's most powerful statistical packages and one of the fastest growing geo-spatial display packages. This paper will demonstrate how SAS® can be used to create files written in Google Earth's Keyhole Markup Language (KML) as well as how these KML files can utilize SAS/IntrNet® to display data dynamically in an easy, user-friendly fashion.

INTRODUCTION

This paper will cover how to use SAS to create KML files for displaying point and non-point (county-level) data. It will also demonstrate how to take advantage of Google Earth's 3-D capabilities for visualizing the data. Finally, and perhaps most importantly, it will describe how to link the KML placemarks to a database using SAS/Intrnet, effectively providing dynamic data access from Google Earth.

BODY

SETTING THE STAGE

Before discussing the "nuts and bolts" of how this application works, we would like to point out that the end user needs only one software package – a licensed copy of Google Earth. The data connectivity happens "behind the scenes" and is completely transparent to the end user. It is part of the simplicity of this application. For it to work, however, a developer must do a little work in advance to set up this process. Specifically, the following components are required:

- A KML file that points to a SAS/IntrNet service
- A working SAS/IntrNet service with connectivity to the source data
- A SAS program that can access the source data and generate the desired output

Once these components are in place, anyone can use the KML file to click on locations and access data dynamically. Figure 1 provides a complete flow chart.

WHAT IS KML?

Google Earth reads files that are written in KML much like a web browser reads files written in HTML. KML is a mark-up language that uses tags to describe elements. There is a "placemark" tag for each point you want to plot. Among other things, these tags allow you to specify where the placemark icons are plotted. The tags allow you to control shapes, sizes, and colors of the placemark icons. And perhaps most important and relevant to the topic of this paper, they allow you to place images in the placemark's description balloon (see Figure 2 for a terminology diagram).

Since you can link to a static image inside the description balloon, there's no reason you can't link to a dynamic image inside the balloons also. Herein lies the key to linking Google Earth with SAS. This is explained further in the "Linking to SAS and source data" section.

MAPPING POINT DATA

BUILDING THE KML FILE

So how do you build a KML file? If you have a lot of time on your hands, you can start from scratch and code the file one line at a time. Another way is to use a software package that can help put your data into KML. Since most of us do not have this much time on our hands and this is a paper about SAS and Google Earth, we will use SAS to do all of this tedious work. The first step is to have SAS write the KML header and footer.

```
DATA _NULL_;
     FILE _webout;
     SET DATA nobs=numobs end=last;
          IF _n_=1 THEN DO;

    rc = appsrv_header ('Content-Type','application/vnd.google-earth.kml+xml');
    rc = appsrv_header ('Content-Disposition',"attachment; filename =
                         &filename..kml");

                PUT '<?xml version="1.0" encoding="UTF-8"?>';
                PUT '<kml xmlns="http://earth.google.com/kml/2.2">';
                PUT '<Document>';
                PUT '<Style id="A">';
                PUT '<IconStyle>';
                PUT '<scale>0.5</scale>';
                PUT '<Icon>';
                PUT "<href>http://webaddress_for_icon.png</href>";
                PUT '</Icon>';
                PUT '</IconStyle>';
                PUT '</Style>';
          END;
```

The "rc = appsrv_header" function is used only if the KML files are to be generated online through the use of a web browser. The first call redefines the headers instructing the web browser to use Google Earth for opening the file. The second call instructs the web browser that this file is for download and gives the option of naming the file. If the KML files are created locally these two lines are not needed. The remaining portion of code relies heavily on the use of the put statement in combination with the basic knowledge of Google Earth's KML code. A great start-up guide for writing KML is available online at http://code.google.com/apis/kml/documentation/ .

The next step in creating the KML files is to plot points stored in the data. The two variables necessary for creating KML placemarks are latitude and longitude. With these key elements, you are now prepared to write the data out a KML file. After defining the header, the next section of code will write out a placemark tag for each observation in the data set. This efficiency is the primary reason for directing SAS to write these files instead of coding them by hand.

```
PUT '<Placemark>';
PUT '<description>';
PUT '<![CDATA[';
PUT 'any information desired to be put in description balloon is placed here';
PUT ']]>';
PUT '</description>';
PUT '<styleUrl>#A</styleUrl>';
PUT '<Point>';
PUT '<extrude>1</extrude>';
PUT '<altitudeMode>relativeToGround</altitudeMode>';
PUT '<coordinates>'||trim(left(longitude))||','||trim(left(latitude))||',0 ';
PUT '</coordinates>';
PUT '</Point>';
PUT '</Placemark>';
```

In KML, the “<![CDATA[“ tag informs Google Earth that information following this tag is HTML code. This ability increases the flexibility of how data are to be displayed and provides more options of how to output the data. The “<styleUrl>” tag is specific to KML and is an icon reference. In the sample code above, the ”<styleUrl>” tag references the icon “A” which was defined in the first step with a “<style>” tag. This method of defining icons and referencing them later is a nice efficiency. It also allows you to define multiple icons that can be referenced conditionally. For example, if you want power plants to be red dots and petroleum refineries to be blue squares, you can use if-then statements to write out the appropriate icon reference based on the type of facility. The final step for creating KML files is to write the end tags. As with the header this portion of code, this needs to be written only once.

```
IF last THEN DO;
    PUT ‘</Document>’;
    PUT ‘</kml>’;
END;
```

With these three segments of code, you now have the ability to output a kml file with placemarks that can be viewed in Google Earth. An optional piece of code can be used to plot the placemarks so they are viewable in 3-D. This is done by scaling the height of the placemark icon based on the data. For example, we can make the facilities’ placemark icon height relative to the emissions they produce. The larger the emissions, the farther the placemark icon is from the ground. This is done in the “<coordinates>” tag. The key elements for this tag are longitude, latitude and elevation. To make use of this feature, you simply have to decide which variable you wish to have the elevation based upon and then insert that variable into the code as shown below.

```
    PUT ‘<Point>’;
    PUT ‘<extrude>1</extrude>’;
    PUT ‘<altitudeMode>relativeToGround</altitudeMode>’;
    PUT ‘<coordinates>’;
    PUT
TRIM(LEFT(longitude))||’,’||TRIM(LEFT(latitude))||’,’||TRIM(LEFT(elevation))||’;
    PUT ‘</coordinates>’;
    PUT </Point>’;
```

What we have described so far is how to use SAS to *create* a KML file. The following section will show how to *link* the KML file to a database using SAS/IntrNet.

LINKING TO SAS AND SOURCE DATA

The key to linking to SAS and ultimately the source data is one simple piece of code that goes in the placemark tag in the KML file. For this to work you must already have a SAS/IntrNet service available (that’s a topic for another paper). When creating KML files, this piece of code is inserted into the “<description>” tag. The code, instead of pointing to a static jpg or gif file, points to a SAS/IntrNet service. The code also identifies the SAS program and the name-value pairs to be passed to the SAS program. The code that was discussed earlier puts a distinct identifier in each placemark tag. This identifier may be character or numeric and is strictly up to how you would like these placemarks to be setup. In this example, we use facility code, or “faci_code”. Now, when the end user clicks a particular placemark icon in Google Earth, the distinct facility code is passed to the SAS program.

```
link='<img src="http://www.epa.gov/cgi-bin/broker?_service=data'||
    '&_program=dataprog.Name_of_SAS_plotting_prog.sas'||
                                '&year1='||"&year1"||
                                '&year2='||"&year2"||
                                '&debug=0'||
                                '&FaCo='||TRIM(LEFT(faci_code))||
                                '"width="460" height="360">';
PUT ‘<Placemark>’;
PUT ‘<description>’;
PUT ‘<![CDATA[‘;
PUT link;
PUT ‘]]>’;
PUT ‘</description></Placemark>’;
```

Because the "<![CDATA[" tag is in the "<description>" tag, you have the power to put HTML code within the description balloons. This allows you to use an HTML image tag to direct Google Earth to a SAS service (as opposed to a static image). This linking provides the ability to allow each placemark icon to have its own image created dynamically. These dynamic images provide a number of advantages including easy maintenance and less storage space for a given KML file.

WHAT HAPPENS IN THE SAS PROGRAM

Once the graphic is generated, it is displayed in the description balloon. How long it takes for the graphic to appear in the description balloon depends on several factors (e.g. amount of data queried, number and type of data processes, server performance). If you are accessing a reasonably small amount of data, the graphic should appear within seconds and in many cases almost instantaneously when the placemark icon is clicked.

Figure 3 shows the type of plot that is generated and displayed when one of the placemark icons is clicked. The plot provides the name of the facility and how many tons of sulfur dioxide and nitrogen oxides (and other pollutants) were emitted in 2002. This is just an example of the type of output that can be provided. Since the output is coded in SAS, the developer of the SAS code can control how it is displayed.

MAPPING COUNTY-LEVEL DATA

Now that you know how to create KML files with *point specific* placemarks, let's discuss how to map *non-point* data in Google Earth. The following section shows how to map data that is aggregated to a county level as seen in Figure 4. The KML code requires a few more tags and development of these maps requires more data preparation than plotting points in Google Earth.

DATA REQUIREMENTS

There are minimal requirements for coloring counties in Google Earth. The data need to contain state and county fips codes or state and county names (if the data contain only one set, the other may be retrieved from the SAS maps library although the first scenario is preferred). The first step is to determine levels and associated colors for the counties. Then you must decide which variable you wish to map. Once all of these steps are in place, what remains is to conduct any type of analysis desired and then to create the KML file.

ASSIGNING LEVELS AND COLOR

This portion is entirely dependent upon how you wish to display the data and there is virtually no restriction on the limits you may set. A PROC MEANS or PROC UNIVARIATE procedure can help you choose cut points that can be stored in macro variables for later use.

```
PROC UNIVARIATE DATA = USmap NOPRINT;
    VAR dense;
    OUTPUT OUT = USmap1 PCTLPTS=30 60 90 PCTLEPRE=P;
RUN;QUIT;

DATA _NULL_;
    SET USmap1;
    CALL SYMPUT('first',P30);
    CALL SYMPUT('second',P60);
    CALL SYMPUT('third',P90);
RUN;QUIT;
```

Once you have stored the cut points into macro variables, you need to create a variable to store the color codes for each observation. The color code combinations used in Google Earth are very similar to standard hexadecimal codes, with a couple of minor differences. In Google Earth, the codes are arranged in the order *aabbggrr*, while standard hexadecimal codes are *rrggbb*. The *aa* combination in Google Earth determines the opacity of the color,

with 00 being completely transparent and FF completely opaque. The remaining segment of the Google Earth color codes is identical to hexadecimal code, except the order which blue, green, and red are placed in the code. One thing to remember is that Google Earth supports only web safe hexadecimal codes.

One simple method for assigning these color codes is to store the codes in macro variables early on in the code to provide quick access and changes when necessary. Creating variables and assigning these codes is as easy as applying simple conditions based on the data. In this example the colors are assigned according to the emissions density for each of the counties throughout the United States.

```
DATA USmap2;
    SET USmap;
    LENGTH one 8 two 8 three 8;
    one = &first;
    two = &second;
    three = &third;
           IF dense >= three THEN DO;
                  color = 'c0000000';END;  *Top 10 % color is black for pollutants
           IF <= dense < three THEN DO;
                  color = &color1;END; *60%-90%;
           IF <= dense < two THEN DO;
                  &color2;END;        *30%-60%;
           IF dense < one THEN DO;
                  color = &color3;END; *0 – 30%;
RUN;QUIT;;
```

RETRIEVING THE COUNTY PERIMETER

Now that your data set contains a new variable to hold the color code, you can move on to the more difficult task of coloring the counties in Google Earth. The method of coloring the counties in Google Earth requires drawing an individual polygon tracing over the perimeter of each county and applying these as overlays on the map. Luckily, the perimeter coordinates are stored in the SAS maps.counties data set. Although you are provided with these coordinates, SAS plots these clockwise while Google Earth only recognizes these coordinates if they are listed counter-clockwise. In addition, they are in radians in maps.counties and Google Earth only recognizes these in degrees. Now that you have these vital pieces of information you are ready to arrange these data into a usable format recognizable in Google Earth. You can perform either procedure first, but we will convert radians into degrees first in this example.

```
DATA county;
    SET maps.counties;
    IF state ^= &stfips THEN DO;
           delete;END;
    IF density > 4 THEN DO;    *remove high density for lower resolution map;
           DELETE;END;
    IF segment > 1 THEN DO;    *remove multiple segments;
           DELETE;END;
    IF x = '.' OR y = '.' THEN DO;  *remove missing data in maps.counties;
           DELETE;END;

*Convert radian lat / long to degrees;

    pi = gamma(0.5)**2;
    long = x * -(180/pi);
    lat = y * (180/pi);
RUN;QUIT;
```

For this example, data was retrieved for a specific state determined by the state fips code (a list of codes can be found online). To reduce the size of the KML file, we have removed some of the coordinates to produce a lower resolution map. The segment portion is designed to map around bodies of water or with unique boundaries. For this

example, all segments were removed to allow for the simplest scenario and the least amount of conditions. Missing values have to be removed since they are not recognized within Google Earth and only produce misshaped polygons.

After converting all of the coordinates into degrees, it is now time to reverse the order of the coordinates from clockwise to counter-clockwise. Even with thousands of coordinates for just a single state, this process is actually easier than it sounds. Before actually beginning to reverse the order of the coordinates, you need to merge the coordinate data set with the data set created earlier containing the county color codes. This is where the state and county fips codes are necessary. We will merge these data together using the county fips so that the correct coordinates are assigned to their proper county contained in our original data. To understand this example, the merged data set is called "merged".

```
DATA count;
    SET merged;
    BY county;
    IF first.county THEN counter = 0 AND num = 0;
    counter+1;
    IF last.county THEN num = counter;
RUN;QUIT;

PROC SORT DATA = count;
    BY cofips descending counter;
RUN;QUIT;
```

The next step is to rearrange the data into a format which will enable you to print to the KML file as described earlier in this paper. The data set "count" actually includes multiple records of the same county with a separate row for each individual coordinate. We would like this data to have only one row for each of the counties and all of the perimeter coordinates listed out in counter-clockwise order amongst the columns. A simple procedure to arrange these data into this format is PROC TRANSPOSE.

```
PROC TRANSPOSE DATA = count OUT=count1 PREFIX = long;
    BY county_name cofips color;
    VAR long;
RUN;QUIT;

PROC TRANSPOSE DATA = count OUT=count2 PREFIX = lat;
    BY county_name cofips color;
    VAR lat;
RUN;QUIT;

DATA latlong;
    MERGE count1 count2;
    BY county_name;
RUN;QUIT;
```

The TRANSPOSE procedure was performed two times to ensure the data were shifted properly for both the latitudes and longitudes and then merged back together. If you look at the data, you will notice that the lat and long variables each contain a numerical suffix. This feature of PROC TRANSPOSE enables you to store each of these perimeter coordinates into SAS arrays and provides a simple method of accessing these data when creating the KML file.

PERIMETERS IN ARRAYS

In this example, the arrays are named according to a segment of the county's name. Since there are not many states with fewer than 20 counties, we are going to have SAS write these definitions to save on programming time. Consider the condition where multiple counties within a state may have the same letters for the first half of the name and not the latter half. For this condition, since you are taking only a segment of the name, you need to ensure that there will never be a case where the array names are the same. To avoid this scenario, you can substring the first eight characters of the county name. From experience, no two counties within the entire U.S. have the first eight characters of their name identical.

```
DATA _NULL_;
    SET nums;
    FILE 'C:\Documents and Settings\user\Desktop\arrays.txt';
    ifname = 'IF county_name = "'||TRIM(LEFT(county_name))||'" THEN DO;';
    temp = LOWCASE(county_name);
    shrtname = SUBSTR(temp,1,8);
    cut = COMPRESS(shrtname , "." , "s");
    arname = 'ARRAY lo'||TRIM(LEFT(cut))||'('||TRIM(LEFT(num))||') long1-
long'||TRIM(LEFT(num))||';';
    arname2 = 'ARRAY lt'||TRIM(LEFT(cut))||'('||TRIM(LEFT(num))||') lat1-
lat'||TRIM(LEFT(num))||';';
    ends = 'END;';
    PUT ifname;
    PUT arname;
    PUT arname2;
    PUT ends;
RUN;QUIT;
```

Now that the county perimeter coordinates are stored in arrays, you can call any one coordinate for a single county at any given time during the DATA step. Since these coordinates are now arranged in counter-clockwise order they can be placed within Google Earth's "<coordinates>" tag, one-by-one, without worry. Since these coordinates must all be placed within the tag in this fashion, one step at a time, you can use a simple loop to go through each array and output the data one cell at a time. As before, you can have SAS write this portion of code into a text file for later use.

```
DATA _NULL_;
    SET nums;
    FILE 'C:\Documents and Settings\user\Desktop\dos.txt';
    temp = LOWCASE(county_name);
    name = SUBSTR(temp , 1, 8);
    cut = COMPRESS(temp, "." , "s");
    name2 = 'IF county_name = "||TRIM(LEFT(county_name))||" THEN DO;';
    dos = 'DO i=1 TO '||TRIM(LEFT(num))||';';
    cords = 'cord =
TRIM(LEFT(lo'||TRIM(LEFT(cut))||'(i))||',"||TRIM(LEFT(cut))||'(i))||",3000 ";';
    pcord = 'PUT cord;';
    ends = 'END;END;';
    PUT name2;
    PUT dos;
    PUT cords;
    PUT pcord;
    PUT ends;
RUN;QUIT;
```

Now you have two files containing the necessary array definition codes and their corresponding DO statements. Above, the "3000" placed within the elevation segment of the "<coordinates>" tag is a necessary requirement if you choose to take advantage of the polygon transparency option within Google Earth. This option will enable a user to view the ground through the polygons.

COLOR THE COUNTIES

Now that you have satisfied all of the necessary steps, you are ready to create county colored overlays in Google Earth. The initial setup is very similar to developing KML files for individual points. First, begin by defining arrays using the code written to the file *arrays.txt* earlier.

```
DATA _NULL_;
    SET latlong end=last;
    FILE <file_ref>;
    *Copy and paste array definitions written to text file here.;
```

Immediately following this initial setup, you have to define the headers for Google Earth. These headers are strictly for Google Earth and are only written using PUT statements. We will change the order in which certain tags for Google Earth will be written to the file only to match the nature of the data (primarily this involves the "<Style id>" tag). We will define these tags based on the county fips and will use the data this time to define the styles as opposed to plotting points where we used a single "<Style id = "A">".

```
IF _n_ = 1 THEN DO;
    PUT '<?xml version= "1.0" encoding = "UTF-8"?>';
    PUT '<kml xmlns = "http://earth.google.com/kml/2.2">';
    PUT '<Document>';
    PUT '<name>NAME OF DOCUMENT</name>';
END;
    style = '<Style id="exNAME'||TRIM(LEFT(cofips))||'">';
    PUT style;
```

Immediately following this portion of code is the introduction of the "<PolyStyle>" tag. This tag is nested within each "<Style id>" tag and is used to define aspects of the polygons. For our example, we will use this tag to define the color of each of the polygons which was determined earlier based on the emissions density for each of the counties.

```
    PUT '<PolyStyle>';
col = '<color>'||TRIM(LEFT(color))||'</color>';
    PUT col;
    PUT '</PolyStyle>';
    PUT '</Style>';
```

Now the remaining portion of the code is very similar to the point plotting example. There will be no descriptions this time, although this option can be implemented. There are a few more new tags added in this section. Based on their names, they should be self explanatory.

```
    PUT '<Placemark>';
    PUT '<name>' county_name '</name>';
styleurl = '<styleUrl>#exNAME'||TRIM(LEFT(cofips))||'</styleUrl>';
    PUT styleurl;
    PUT '<Polygon>';
    PUT '<altitudeMode>relativeToGround</altitudeMode>';
    PUT '<outerBoundaryIs><LinearRing><coordinates>';

*Paste the DO statements written to text file here.  This writes the lat / long
data
Stored in the arrays to the KML file;

    PUT '</coordinates></LinearRing></outerBoundaryIs></Polygon>';
    PUT '</Placemark>';
IF last THEN DO;
    PUT '</Document>';
    PUT '</kml>';
END;
RUN;QUIT;
```

SPECIAL CASES

When creating county colored KML files there are three special cases which require slight changes in code. These cases include Denver, CO, Miami-Dade, FL and the entire state of Alaska. For Denver, maps.library does not include perimeter coordinates and you have to define this perimeter manually. In Florida, the county officially changed it's name from Dade County to Miami-Dade County in 1997. If you are working with data from 97-99 (1999 is the official change of fips from 25 to 86), then you must check that maps.library and your data set are using identical fips codes. For the state of Alaska, maps.counties does not include some of the more recently distinguished boroughs. Also for

Alaska, maps.counties contains multiple segments for each borough. With the code provided above, these multiple segments are removed and the resulting maps are missing large portions of the borough regions. For each of these cases, the perimeter coordinates will not be stored using arrays and alternatively will be stored using macro variables in a DATA _NULL_ step preceding the final datastep to create the KML file.

```
DATA _NULL_;
    SET latlong;
    IF county_name = "Denver" THEN DO;
    initial = 8;

CALL SYMPUT('denv0' , initial);
    CALL SYMPUT ('denv1' , "-104.7337929,39.812544295,3000 -
104.7876694,39.78331108500001,3000 -104.8649872,39.812544295,3000");
    CALL SYMPUT ('denv2' , "-104.9022527,39.783884821,3000 -
104.9709917,39.80164331300001,3000 -105.0523529,39.790769652,3000 -
105.0523529,39.66758036,3000");
    CALL SYMPUT ('denv3' , "-105.0769962,39.669301567,3000 -
105.0620791,39.645805717,3000 -105.1245343,39.616572507,3000 -
105.0523529,39.609714997,3000");
    CALL SYMPUT ('denv4' , "-105.0523529,39.613130092,3000 -
105.0523529,39.622883602,3000 -105.0523529,39.624031074,3000 -
105.0523529,39.627473489,3000");
    CALL SYMPUT ('denv5' , "-105.0523529,39.629194697,3000 -
105.0523529,39.63148964,3000 -105.0277096,39.628047225,3000 -
105.0334469,39.65323696200001,3000");
    CALL SYMPUT ('denv6' , "-105.0076561,39.678454021,3000 -
105.0099511,39.660121793,3000 -104.9727402,39.66758036,3000 -
104.8838931,39.624604809,3000");
    CALL SYMPUT ('denv7' , "-104.8793032,39.652663227,3000 -
104.8472287,39.658974321,3000 -104.9027991,39.66758036,3000 -
104.907389,39.683617644,3000");
    CALL SYMPUT ('denv8' , "-104.8649872,39.701949872,3000 -
104.8838931,39.739761799,3000 -104.855261,39.768995008,3000 -
104.7337929,39.768995008,3000");END;
RUN;QUIT;
```

The procedure here is simple. There is a separate macro variable to contain a portion of the perimeter coordinates. These coordinates have already been defined in counter-clockwise order and only need to be output to the KML file. When outputting these coordinates, the procedure is slightly different than before.

```
DATA _NULL_;
    SET latlong end = last;
    FILE KML;
    IF county_name = "Denver" THEN DO;
    %DO i=1 %TO &denv0;
           PUT "&&denv&i";
    %END;
END;
RUN;QUIT;
```

For Miami-Dade County you need to use the method just defined or reset the county fips code to match your data with maps.counties. Alaska simply needs to use the method above for each of its boroughs. A file containing Alaska's coordinates has been attached to the document. These coordinates have already been defined in the fashion provided above.

CONCLUSIONS

This application merges two powerful software products and leverages their strengths to create a super-powerful, yet super-simple analysis tool. Because Google Earth is so intuitive, many data providers are beginning to offer their data in KML. This paper provides one example of an application for air quality analysis, but it just scratches the surface. There is great potential for applications like these to be utilized increasingly in the future.

REFERENCES

Google Earth KML documentation on World Wide Web, http://code.google.com/apis/kml/documentation/.

SAS Web Tools: Advanced Dynamic Solutions Using SAS/IntrNet Software Course Notes, SAS Institute Inc., Cary, NC, 2001.

ACKNOWLEDGMENTS

We would like to thank Scott Goodrick (USDA Forest Service) and Joe Abraham (U.S. EPA, Region 9) for help with understanding the inner workings of Google Earth, Michael Rizzo (U.S. EPA, Office of Air Quality Planning and Standards) and Bill Smith (U.S. EPA Office of Environmental Information) for insights regarding KML file-building using SAS software, and Nick Mangus (U.S. EPA, Office of Air Quality Planning and Standards) for actually coming up with the idea of placing SAS results inside the description balloon. We would also like to thank our colleagues on the EPA Emissions Inventory & Data Analysis Team for comments throughout the development process: Doug Solomon, Linda Chappell, Tom Pace, Ron Ryan and Tesh Rao. Finally, we would like to acknowledge Tom Curran who initially challenged us to explore connections between Google Earth and SAS.

DISCLAIMER

This paper represents the views of the authors and not necessarily those of U.S. Environmental Protection Agency (EPA). Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

KEY WORDS

Google Earth
SAS
GIS
Emissions
Air Quality
Data Analysis
Mapping
Spatial Analysis

CONTACT INFORMATION

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

Joshua Drukenbrod
U.S. Environmental Protection Agency
109 T.W. Alexander Drive
Mail Code: C339-02
Research Triangle Park, NC 27711
Work Phone: (919) 541-0928
Fax: (919) 541-0684
E-mail: drukenbrod.josh@epa.gov

David Minz
U.S. Environmental Protection Agency
109 T.W. Alexander Drive
Mail Code: C304-04
Research Triangle Park, NC 27711
Work Phone: (919) 541-5224
Fax: (919) 541-3613
E-Mail: mintz.david@epa.gov

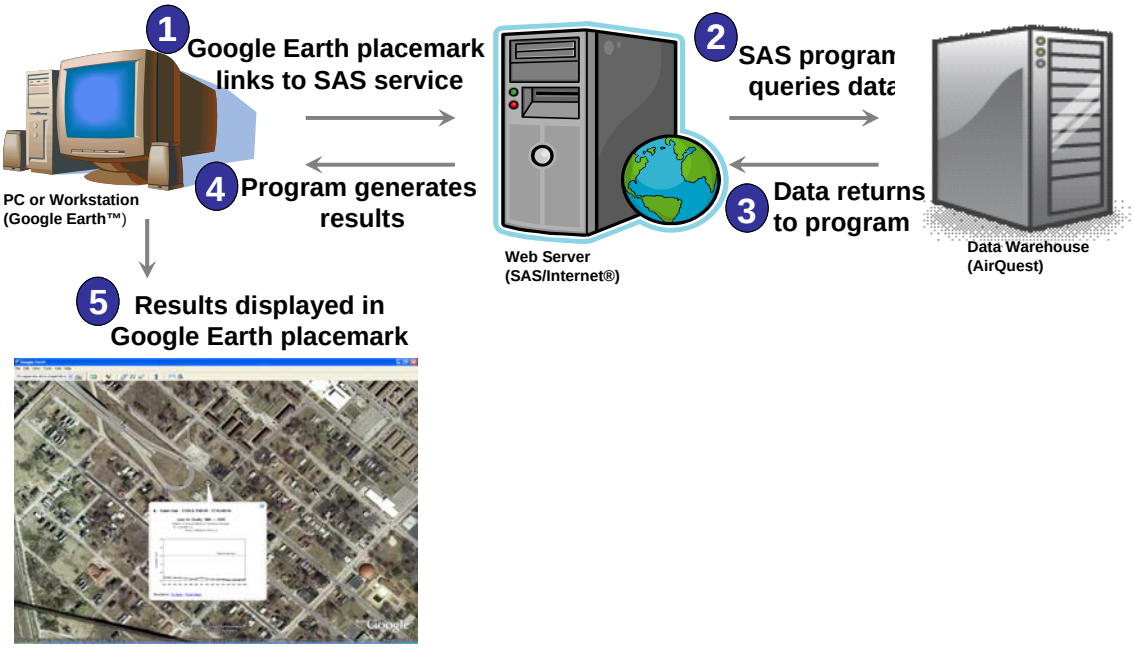

Figure 1. Flow chart explaining how Google Earth can be linked to SAS/InterNet service.

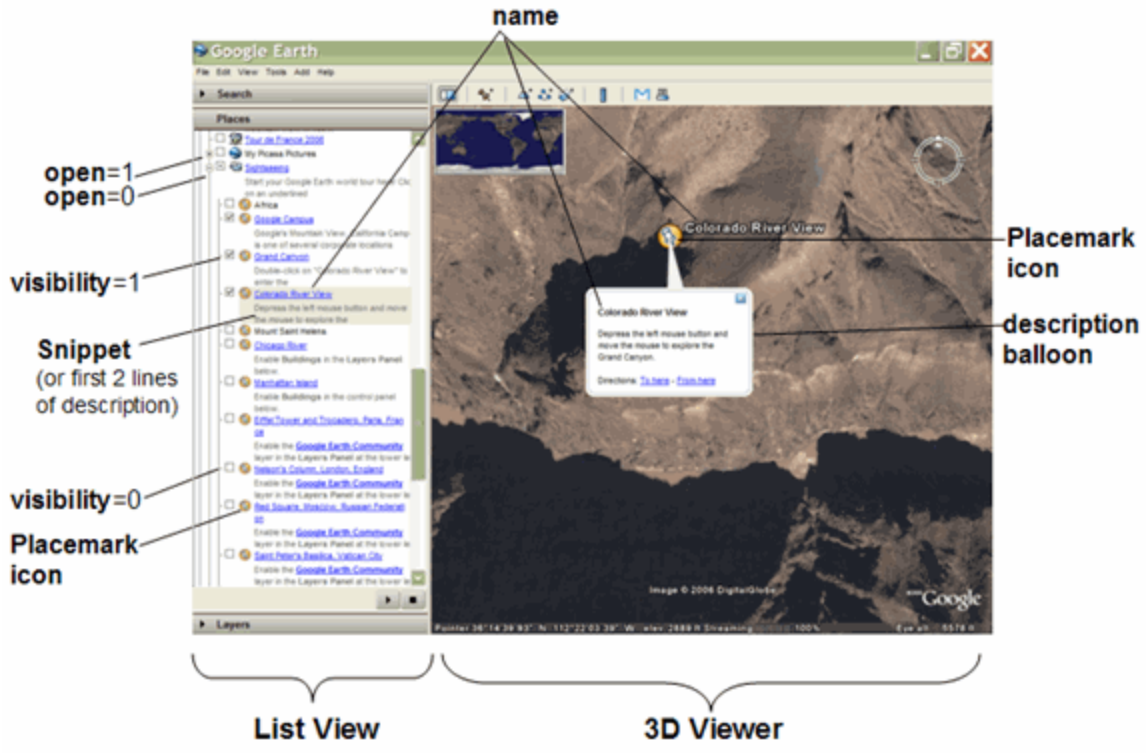

Figure 2. Google Earth terminology and reference diagram. Source: http://code.google.com/apis/kml/documentation/kml_tags_21.html

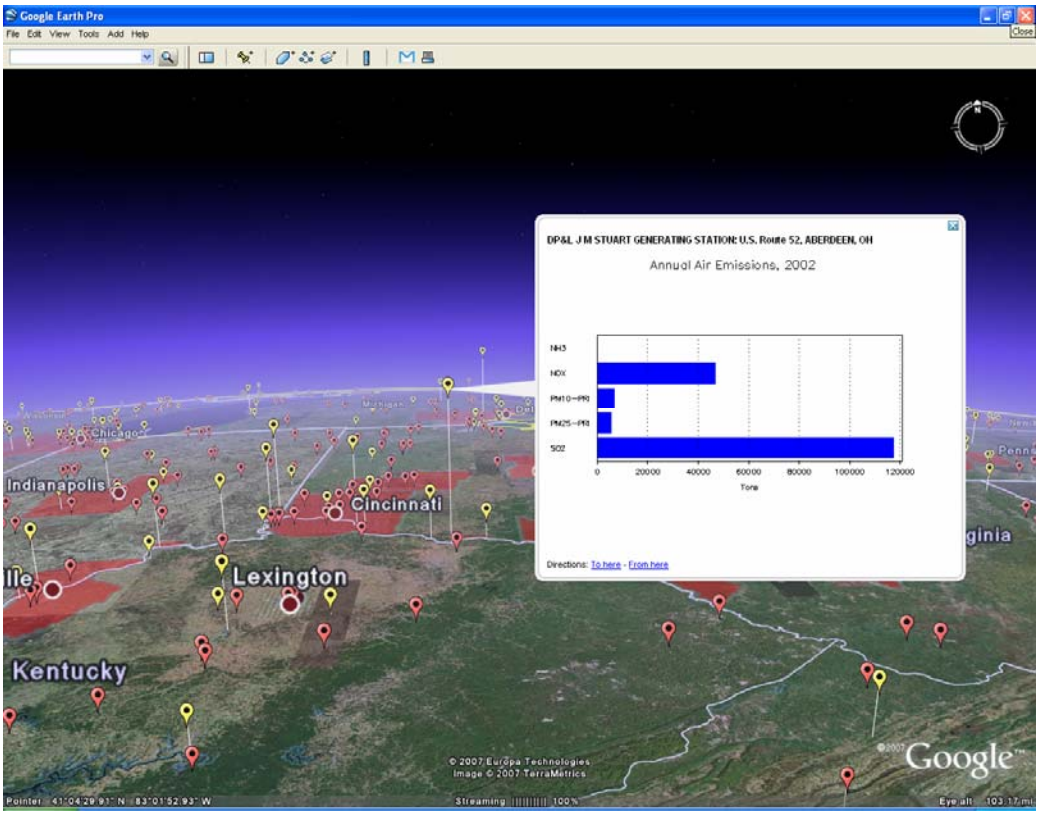

Figure 3. Google Earth displaying KML of emissions point sources, including a description balloon with SAS output.

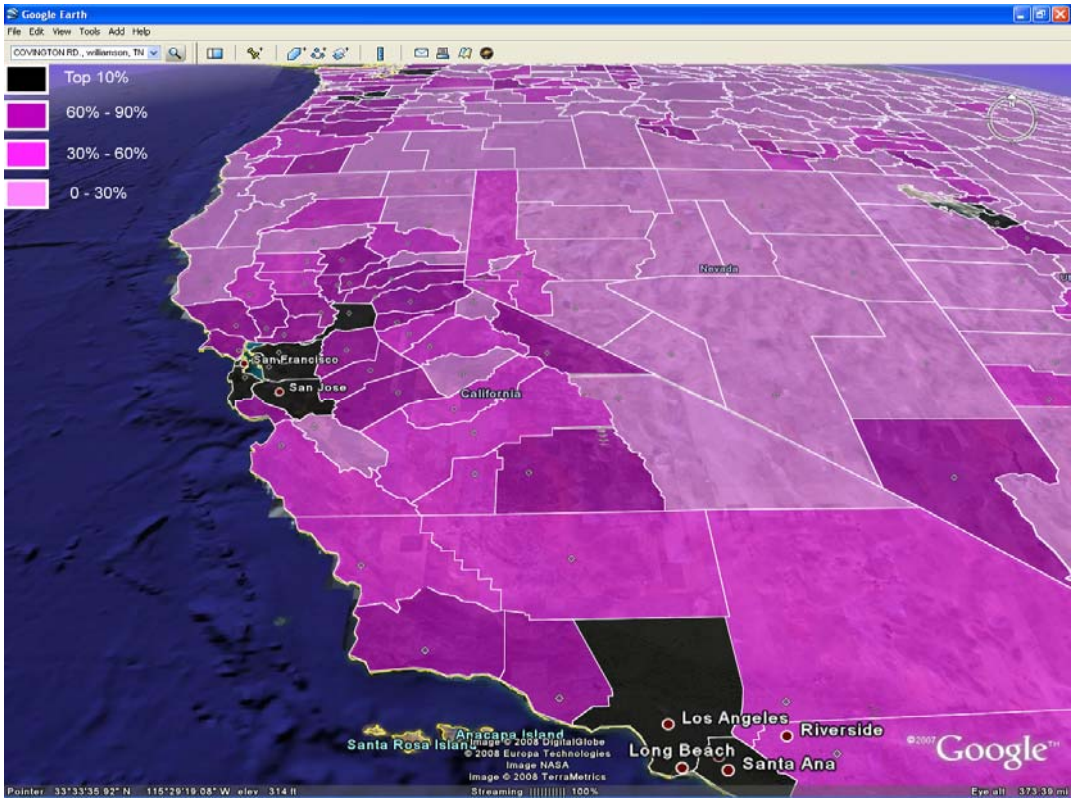

Figure 4. Google Earth displaying county-level emissions densities of VOCs.