Paper 233-2009

Notes from an Intersection: Google Earth @ SAS®
Carol Martell, UNC Highway Safety Research Center, Chapel Hill, NC

ABSTRACT

This paper examines the issues one encounters using SAS to create data-driven placemarks viewable in Google Earth. The introduction of bike lanes in certain areas of St. Petersburg, Florida, provides a scenario for comparing before-and-after traffic counts. The traffic count data, while providing an intersection label, does not provide latitude and longitude. Come along for a ride on the SAS train from proprietary traffic counts to Google Earth; there are sure to be many detours. Is it a rollercoaster or a kiddie ride?

At the 2008 SAS Global Forum, two integrative techniques were unveiled. Kuiper and Vyverman showed us how to use SAS XML Mapper to read, and in SAS 9.2, write Google Earth XML (KML). Drukenbrod and Mintz showed us how to provide dynamic query access at placemarks using SAS/IntrNet® Application Dispatcher and how to create county-area colored overlays for Google Earth. With these techniques in hand, we set out in search of some appropriate ways to display our data.

INTRODUCTION

The casual user of Google Earth is familiar with the process of manually creating a placemark. While it may be alluring to think of visualizing statistical data surrounded by its natural and built environment, the thought of manually creating a quantity of placemarks is hardly enticing. We will examine methods to display SAS data in Google Earth to allow meaningful data exploration. Using the context of the collected traffic count data, this paper discusses problems encountered and solutions employed while transforming SAS data for display in Google Earth. The SAS XML92 LIBNAME engine and XML Mapper are discussed at length. The use of SAS/IntrNet htmSQL is briefly discussed. Helpful tips are preceded by a red double-asterisk (**)

THE DATA

Traffic counters were placed near intersections on roadways both before and after the introduction of bicycle lanes. Each counter was left in place for at least a week in each location, and recorded a timestamp and vehicle type for every vehicle passing over the device. In this paper, we refer to the data collected prior to bike lane installation as 'before' data, and the post-installation collection as 'after' data. Because the recording devices were placed near the curb, once a bicycle lane was installed, heavier vehicles only crossed the device when encroaching into the bike lane. Consequently, only the before heavier vehicle traffic counts are accurate. The heavier vehicles are categorized as light, medium and heavy, representing passenger vehicles, buses, and heavy trucks, respectively. The recorded intersection name ("Central Avenue at 1st Avenue North Bound") provides the means to obtain latitude and longitude for each multi-day observation

THE GOOGLE EARTH KML FILE

KML is an open standard maintained by the Open Geospatial Consortium, Inc. (OGC). The KML elements supported in Google Earth are documented at http://code.google.com/apis/kml/documentation/. A file constructed according to these standards can be displayed in Google Earth. This paper describes the process of using placemarks, lines, styles and icons to construct a KML file that visually reflects our data. We will use the both XML Mapper and htmSQL to create KML files.

A placemark defining a point causes an icon to display on the Google Earth map at a specific geographic location. The coordinates can include altitude in addition to the required latitude and longitude. A line can be extruded from the elevated point to the ground, illustrating the altitude. A line placemark has a series of geographic coordinates in its definition - the points to be joined by the line. Styles specify and scale icons, vary line widths, and modify line and icon colors, as seen in the following examples.

To visualize our data, we have the following plan. We will create a placemark for each vehicle type at each intersection at an altitude that reflects traffic volume for that vehicle type. The placemarks will differ in appearance according to vehicle type. At each intersection, the placemarks will appear adjacent to one another rather than on top

of one another. Placemarks representing before and after data will differ in color. Lines will be constructed from intersection to intersection to illustrate traffic volume comparisons.

We begin by exploring the tools available in KML to accomplish our goals. Here is a sample placemark followed by a screenshot of the placemark in Figure 1. The placemark name 'My Place' labels the icon on the map, entitles the information balloon, and names the placemark in the Places navigation sidebar. Notice the extruded white line from the icon to the ground.

```
<Document name="Paper Samples">
  <Placemark name="My Place">
    <description>Text that appears in the balloon</description>
    <Point extrude="1" altitudeMode="relativeToGround"
     coordinates="-82.675125,27.708944,19.5 " />
  </Placemark>
</Document>
```
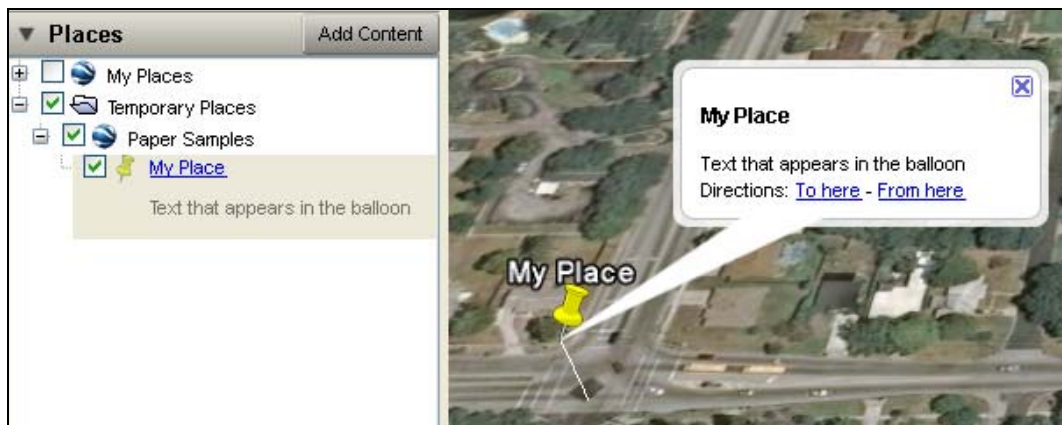

Figure 1 Google Earth view of placemark without styles

We explore customizations that will help us differentiate between data points by defining and applying a style that specifies an icon, colors and line width. We add a reference to this style in our placemark. We also use a description containing **HTML markup to better control the appearance of the balloon text. Figure 2 shows that the line color, line weight, and icon have changed, as has the description inside the balloon.

```
<Style id="BeforeB">
    <LineStyle color="ff00ffff" width="2" />
    <IconStyle color="ff00ffff" scale=".8" >
        <Icon href="http://mywebserver/yellow_circle.png" />
    </IconStyle>
</Style>
<Placemark name="My Place">
  <description><![CDATA[<h3>Text</h3>This text is<br><b>HTML-formatted</b>]]>
  </description>
  <styleUrl>#BeforeB </styleUrl>
  <Point extrude="1" altitudeMode="relativeToGround"
     coordinates="-82.675125,27.708944,19.5 " />
</Placemark>
```
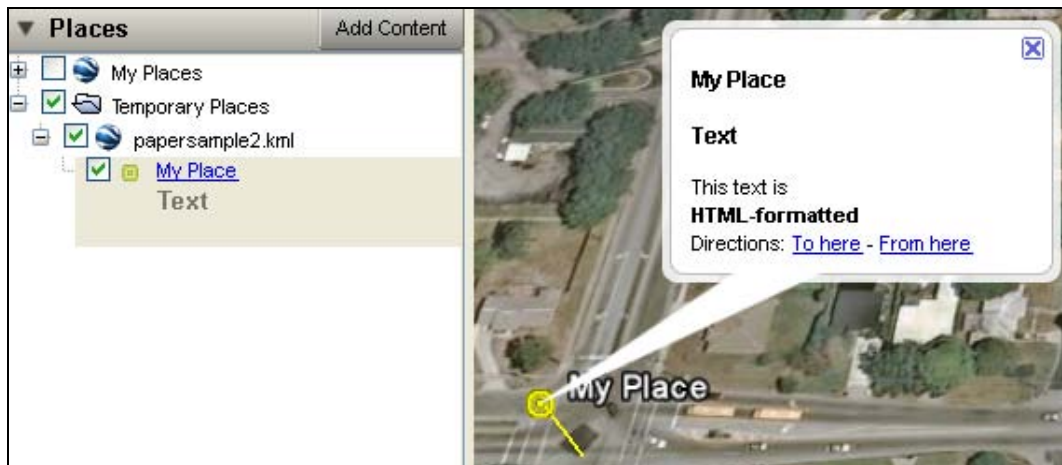
Figure 2 View of Figure 1 placemark with styles and with HTML markup applied

Here is KML markup defining a line, followed by the resulting Google Earth screenshot in Figure 3. The view chosen illustrates that the line is floating above the earth, joining points of differing heights.

```
<Placemark name="A Sample Line">
   <styleUrl>#BeforeB </styleUrl>
   <LineString extrude="0" tessellate="1" altitudeMode="relativeToGround">
      <coordinates>
            -82.675315,27.770093,30.3
            -82.675321,27.770968,18.9
            -82.675330,27.771848,19.0
      </coordinates>
   </LineString>
</Placemark>
```
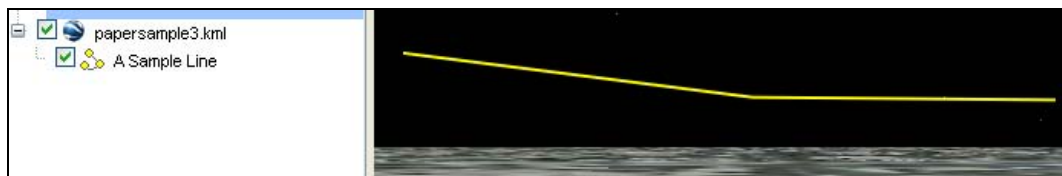

Figure 3 Google Earth view of line joining points of differing altitudes

WRITING KML FROM SAS DATA

We plan to create both point and line KML placemarks from our traffic volume data. We can use PUT statements in a DATA step to generate the markup we've seen above; that approach is not discussed here. Another method involves using the XML92 LIBNAME engine available in SAS 9.2. We discuss and illustrate the use of this LIBNAME engine in great detail. Finally, we examine a very easy method - SAS/IntrNet htmSQL. We briefly discuss this preferred method, assuming the reader is an experienced htmSQL user.

THE SAS 9.2 XML LIBNAME ENGINE

An XMLMap informs SAS of the structure of an XML file. Prior to 9.2, an XMLMap could only be used to read XML data into SAS. SAS 9.2 has the XML92 engine that allows the user to write XML data in a structure defined by an XMLMap. The Java-based XML Mapper ships with SAS 9.2. It is a stand-alone application wherein the user can open an XML file and, using a drag and drop process, create an XMLMap. The map is then referenced in a LIBNAME statement. The process is illustrated in the following section.

XML Mapper – AN EXAMPLE

Our goal is to create an XMLMap that will tell SAS how to clone our sample XML file structure. In this example, we will clone the following very basic placemark.

```
<Document>
 <Placemark name="My Place">
    <Point coordinates="-82.675125,27.708944,19.5 " />
 </Placemark>
</Document>
```

We save the five lines into a file called "basic.kml". **Although it is possible to create an XMLMap from scratch, it is easier to begin with a sample of the KML you desire. **Test the validity of your model KML in Google Earth!

We launch XML Mapper and open "basic.kml" using "File/Open XML". We look at the left application pane. In the Full tab, the data is displayed, and in the Condensed tab, only the structure is displayed, as seen in Figure 4.
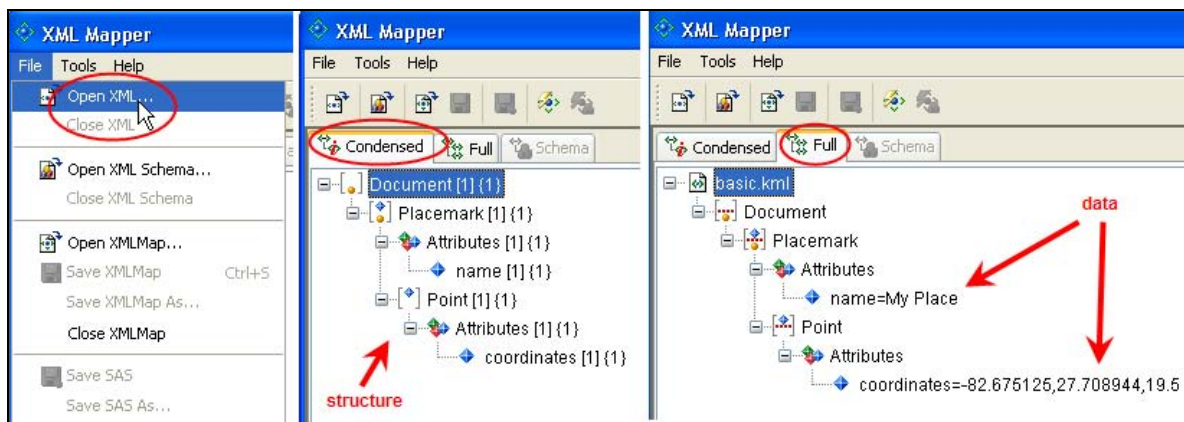

Figure 4 Three views of left pane in XML Mapper when opening "basic.kml"

In the XML Mapper right pane we see an empty map structure. First, we type in the map name "BasicPlacemark". **This name dictates the filename of the output KML file. Next, we drag "Placemark" from the left pane structure and drop it onto "BasicPlacemark" in the right pane. This dictates our output SAS table name. **The SAS table name you use when writing to the xml library must match this name. Next we drag "name" over and drop it onto "Placemark." These steps are illustrated in the series of images in Figure 5. (As of this writing, the SAS version 9.2 xml engine will only write a single table and the map must be version 1.2)
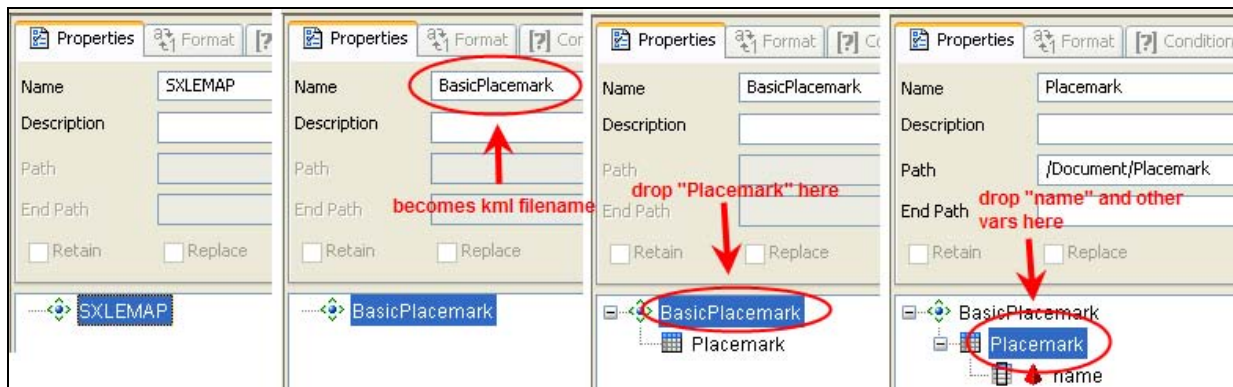

Figure 5 Four views of right pane in XML Mapper when creating the XMLMap

We drag "Point" over and drop it onto "Placemark". We drag "coordinates" over and drop it onto "Placemark" as well. **It may seem counterintuitive to drop the items deep in the nested structure on the left onto "Placemark". It helps to think of this as dragging variables over to drop onto a table. There can only be three levels on the right – the map, the table, and the variables. We will now examine the parameters of each variable in our map to see if they need any changes. **Do not modify the names of the items that you drag over, as these will become the very specific, case-sensitive KML tag names!

Click the Properties tab in the right pane. Highlight, in turn, "Placemark", "name", "Point", and "coordinates". We see this in the series of images in Figure 6. Path values denote XML nesting structure. Notice the highlighted '@' in the path values for "name" and for "coordinates". The '@' indicates that data values will be contained within the opening

tag itself as a name/value pair, rather than nested between a pair of tags. The effect of '@' is illustrated in Figure 7. **Using the name/value pair approach as much as possible minimizes the possibility of bugaboos; test the name/value pairs in Google Earth before using them!
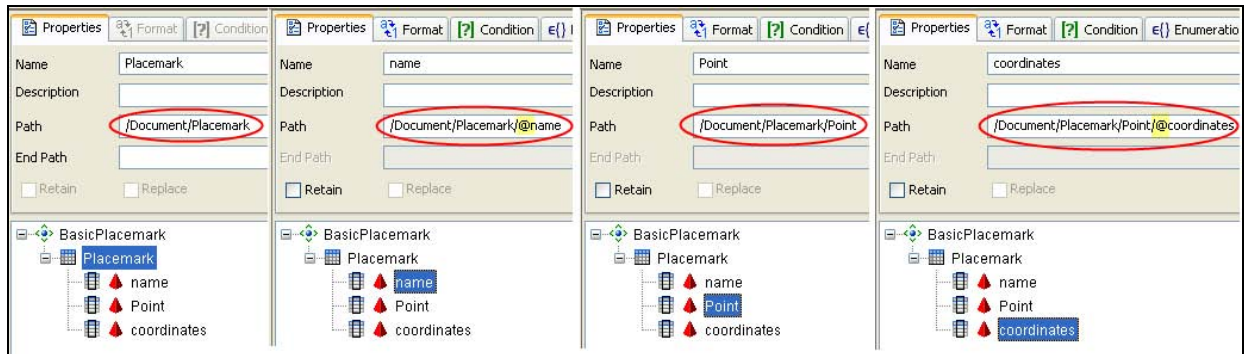

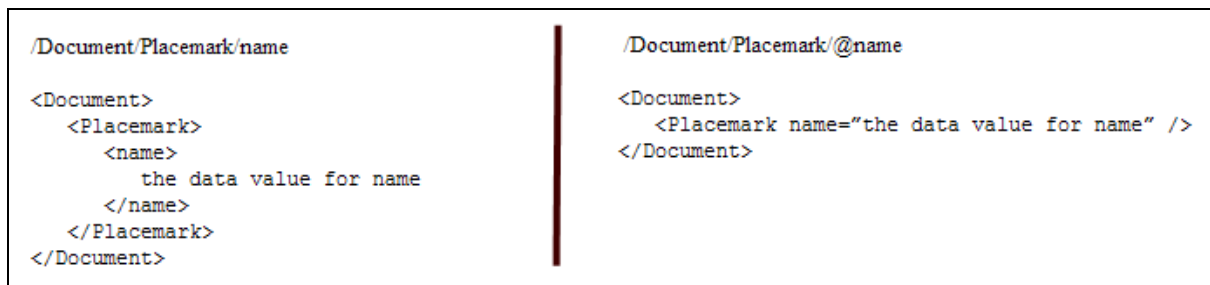**Figure 6　Four views of right pane in XML Mapper when examining the path values**

Figure 7　Comparing XML output results of '@' in a path

Now, click the Format tab and examine the field lengths. Data types and field lengths are derived from the data values found in the template - "basic.kml". **The data types and field lengths are crucial. Later, when using the map, if a SAS character field is longer than what is specified here, SAS will hang, crash, or come back with overflow errors. RESTART SAS if this happens! If your SAS character field is shorter, a warning message is issued, but processing continues. A SAS data type mismatch between input SAS data and the map definition yields an error message, at best.
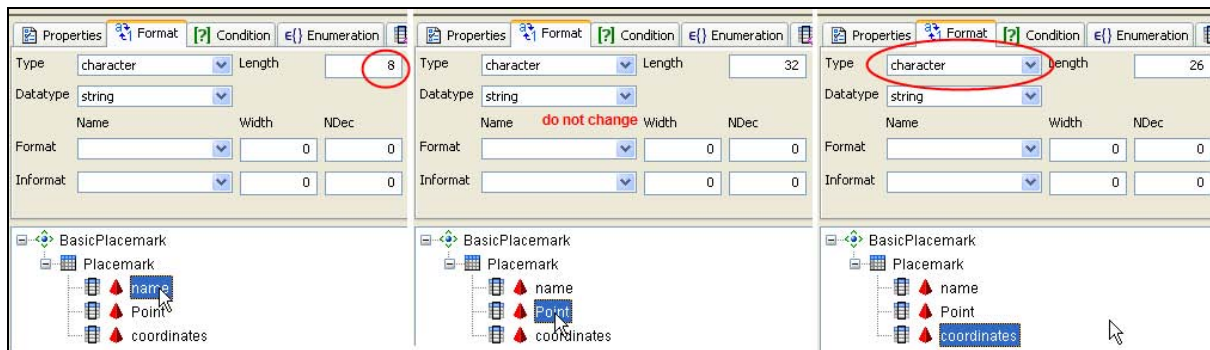

Figure 8　Three views of right pane in XML Mapper examining the assumed types and formats

Figure 8 shows the format screens for making adjustments to the three items under "Placemark" in our map. We change character field length to accommodate our SAS data; the box for setting length is circled in the first pane. We leave "Point" alone, since "Point" is not actually going to correspond to a variable in our SAS data. It is included because "coordinates" is a name/value pair that will appear inside the <Point> tag. We change the field type wherever the type assumption is incorrect; the select list for changing type is circled in the third pane. We save the XML Map

using "File/Save XML As…" from the menu bar. We save it as "BasicPlacemark.map" (not illustrated here.) We will use this SAS XMLMap to create a few placemarks on the SAS Campus in Cary, NC.

We create a SAS table with variable names matching those found in the map. We issue filerefs for the XMLMap and for the output KML file; we also issue a libref for the output KML file. The LIBNAME statement points to the map fileref and **the libref name must match the KML fileref name. **The DATA step must write a table named "Placemark", because that is the name we used in the XMLMap. Figure 9 shows the results in Google Earth.

```
DATA sasplaces;
name="Entrance"; coordinates="-78.766311,35.830543"; OUTPUT;
name="Guards"; coordinates="-78.764131,35.828149"; OUTPUT;
name="Umstead"; coordinates="-78.764256,35.829937"; OUTPUT;
RUN;
FILENAME kmlout "…filepath\BasicPlacemark.kml";
FILENAME mymap "…filepath\BasicPlacemark.map";
LIBNAME kmlout XML92 XMLTYPE=XMLMAP XMLMAP=mymap;
DATA kmlout.Placemark; SET sasplaces; RUN;
```


Figure 9 Our placemarks on the SAS campus in Cary

APPLYING THE TECHNIQUES TO OUR DATA

The above example illustrates the process of generating KML from a SAS table by using the XML92 engine with an XMLMap created in XML Mapper. We now proceed to create the more complicated KML files that we need for our study data. We need to design our placemarks, create XMLMaps that will generate those placemarks, and create SAS tables that can use the XMLMaps. For our data, we need one record per placemark having map coordinates for the intersection and variables named and typed to match our XMLMap.

XML MAPPER – MAPS FOR OUR DATA

```
<Placemark name="" styleUrl="#LB">
    <description> Pinellas Point Dr S</description>
    <Point extrude="1" coordinates="-82.675145,27.708944,295.3"
           altitudeMode="relativeToGround" />
    <LookAt longitude="-82.675125" latitude="27.708444"
           range="100" tilt="45" heading="30" />
</Placemark>
```

The above KML placemark markup is our template for SAS XMLMapper to read as a guide for creating a vehicle-type placemark. Notice the blank name field. If every placemark is given a "name" value that includes the vehicle type and a "description" value containing all of the statistics for the intersection (hours, rates, counts, etc.,) we have a problem. When all the placemarks representing different vehicle types are displayed together for comparison, we see a cluttered map like that shown in the left pane of Figure 10. Every icon has a label displayed. This would not be a

problem if the icons were not in close proximity, but we need them close together for visual comparison. For this data, it is equally disappointing to see all the vehicle type icons mapped to the exact same point, varying only in altitude, as seen in the right pane of Figure 10. We prefer that the vehicle placemarks appear side by side.

Figure 10 Problems: too many labels obscure and clutter a map; placemarks varying only in elevation

For our data, since the icons indicate the vehicle types and colors denote the study group for the bicycles, there is no need for a label to display adjacent to each vehicle icon. **Our de-cluttering solution is to create an extra set of icons to label the intersection and also provide balloon statistics, leaving the vehicle icons in a sort of verbally-minimalist state.

To eliminate a label on the map, we assign a blank name field (we could simply leave it out). In the sidebar navigation, however, we want to be able to find each specific intersection for the vehicle type. If we provide neither "name" nor "description" values, the sidebar navigation reveals a list of unlabeled icons. If an icon contains a value for "description", the value displays beneath the icon name in Google Earth. We use this feature to display identification in the sidebar navigation. Figure 11 illustrates enhancement a description can provide for an unnamed placemark.

```
<Placemark styleUrl="#BA"><Point coordinates="-82.675135,27.708944" />
   <description></description>
</Placemark>
<Placemark styleUrl="#BA"> <Point coordinates="-82.675174,27.719170" />
   <description>Description value</description>
</Placemark>
```


Figure 11. Two unnamed placemarks

Here we see KML for an intersection placemark. Figure 12 shows some details of the XMLMap we create by reading this placemark template in XML Mapper, and dragging and dropping to create our XMLMap.

```
<Placemark name="31 St @ Pinellas Point Dr S" styleUrl="#I">
   <description><![CDATA[Hours: 1,402 before, 1,273 after<br>Bikes: 644 before, 537
    after<br>Bikes/day: 11 before, 10 after<br>17,253 light vehicles for
    5/day<br>623 medium vehicles for 11/day<br>80 heavy vehicles, 1/day<table>]]>
   </description>
   <Point coordinates="-82.675125,27.708944" />
   <LookAt longitude="-82.675125" latitude=" 27.708444"
          range="100" tilt="45" heading="30" />
</Placemark>
```
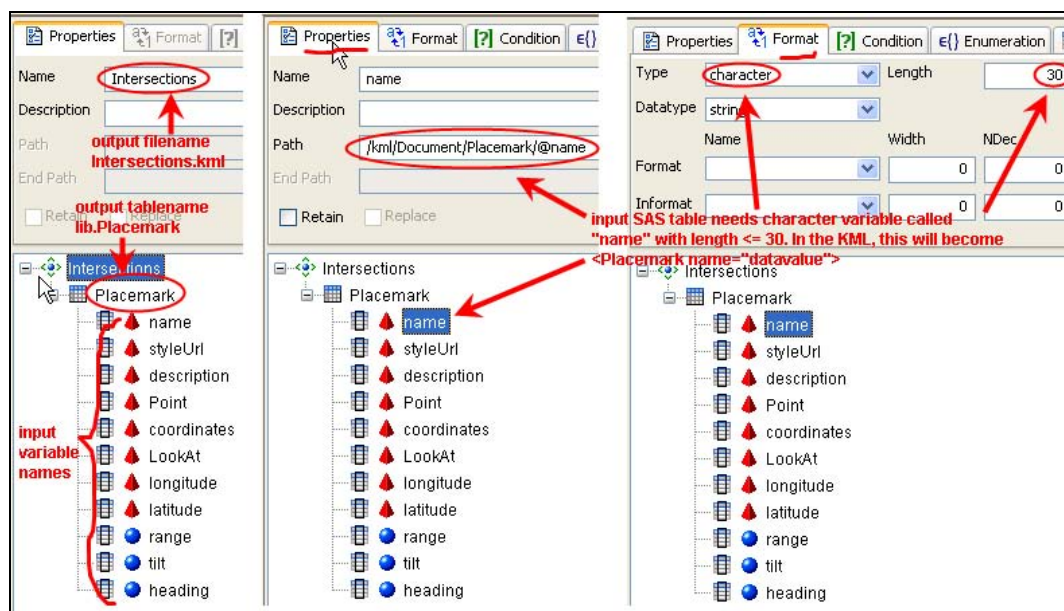
7

Figure 12 The intersection XMLMap showing crucial input data and map correlations

We also will need a map to display our lines traversing intersection points for similar vehicle types. A line is a single placemark wherein the points to join are supplied as a series of geographic coordinates separated by spaces. Setting the LineString parameter "tessellate" to 1 causes the line to be drawn from point to point on the map.

```
<Placemark styleUrl="#BA" name="37 St Corridor">
   <LineString coordinates="
    -82.683388,27.748321,12.6 -82.683435,27.751942,16.4 -82.683473,27.755522,20.0
    -82.683450,27.766402,17.1 -82.683428,27.769998,12.7 -82.683457,27.770952,11.5
    -82.683473,27.771829,12.7 -82.683502,27.777311,17.9 -82.683525,27.780878,11.7"
    tessellate="1" altitudeMode="relativeToGround" extrude="0" />
   <LookAt longitude="-82.693449" latitude=" 27.725906"
           range="3300" tilt="65" heading="45" />
</Placemark>
```

Using the above as an input template for XML Mapper, we create the XMLMap seen in the left pane of Figure 13. Notice that the length for the character variable "coordinates" is set at 500. XML Mapper had set this length in the 200 range, using the template data value to determine length for the variable. We intend to join more than the nine points shown in the KML input template, so we increase the value.

For our vehicle placemarks, we have one map structure that applies to all the vehicle types. Although the structure is the same, varying styles will cause them to look different on the map. The data values for "styleUrl" will dictate the appearance, which will be different for each vehicle type.

We want five different output KML files so that we can easily turn the display of each vehicle type on and off as we examine the study data in Google Earth. We create five XMLMaps that differ only in map name. We do this because the map name must match the output KML filename. We could use one map and rename the output KML file between iterations to avoid over-writing output files or we could use one map and write each identically-named KML output file to a different folder. For illustrative purposes, we simply iteratively rename and save the map for each vehicle type. We name the map "BikesBefore" and save the map as "BikesBefore.map", then replace the name with "BikesAfter" and save it as "BikesAfter.map." This process is illustrated in the right three panes of Figure 13.
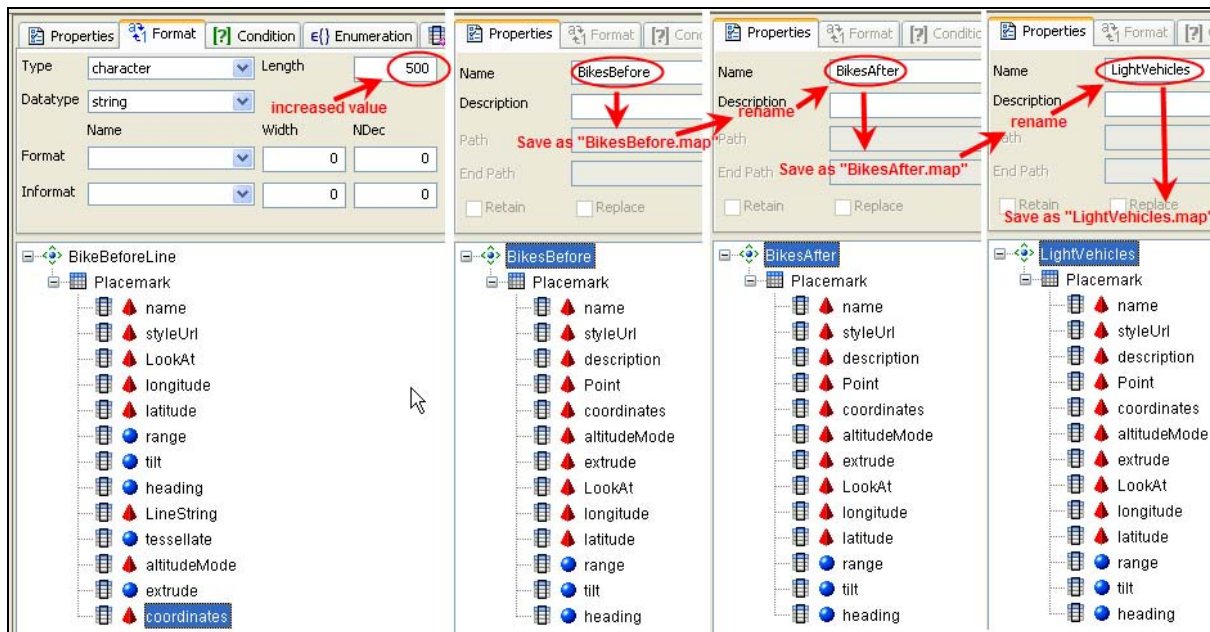
8

Figure 13 Left pane: increasing field length for line placemarks. Right three panes: saving vehicle type maps

Now that we have created the XMLMaps to generate the three types of placemarks, we need data conforming to the input requirements of these maps. We need to position ourselves to be able to pair a SAS table having one record per intersection with the appropriate XMLMap. We will pair a SAS table having one record per intersection with the light vehicle XMLMap, another SAS table having one record per intersection with the heavy vehicle XMLMap, and so forth.

PREPARING THE DATA

GEOLOCATING

Free tools are available to obtain latitude and longitude from an address, for example www.batchgeocode.com. It is also possible to use a Google Earth API for address lookup - the method used to geolocate this intersection data - http://code.google.com/apis/maps/documentation/services.html#ReverseGeocoding. The details of the process are not discussed herein. We obtain coordinates for our intersections, storing them in the SAS table "out.addresses".

ICONS

Since we wish to display points representing different things in proximity to one another, we'd like to use more than just color to indicate what each point represents. We modify icons to represent the vehicle types, coloring the 'before' icons yellow, and the 'after' white, as seen in Figure 14. We place the icons on a web server so that we can refer to them in style definitions. To use icons from the set available in Google Earth, find the URL address for the icon by choosing "Add/Placemark" from the Google Earth menus or by clicking the Add Placemark icon in the Google Earth toolbar. Next click the pushpin icon in the "New Placemark" window, as seen in Figure 15. Icon selections will display along with a URL for the currently selected icon, as seen in Figure 16.
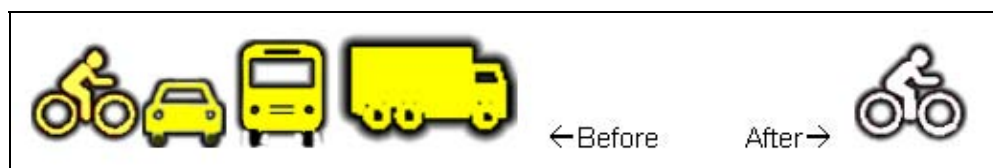

Figure 14 Custom icons created to reflect vehicle type and study group
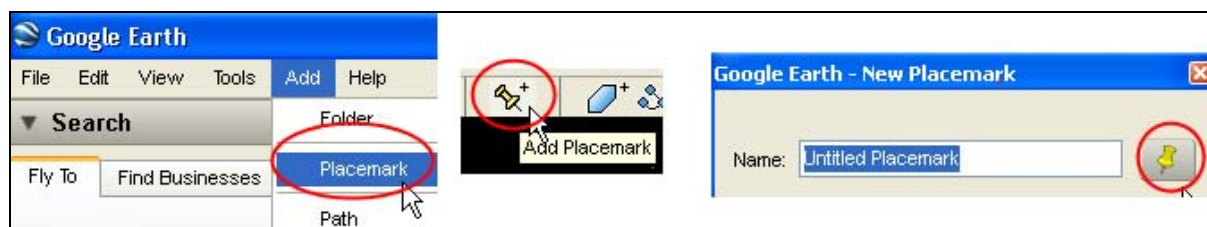
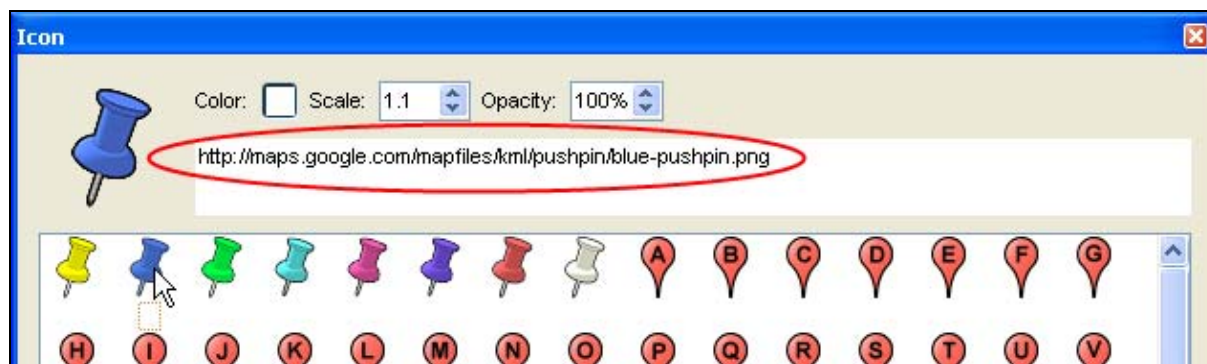**Figure 15 Finding the Google Earth icons**

Figure 16 Obtaining URL for Google Earth icon

STYLES

In order to use the various icons we have created, we use KML markup to create a style for each vehicle type. In addition to supplying the URL for the icon, we also specify a color for lines, a width for lines, and a scaling factor for the icon. The example shown below has the id "BA". To use this style in a placemark, we assign the value "#BA" to the variable "styleUrl" in every observation representing after bicycle rates. The use of "#" in front of the "id" value is similar to the "#" usage in HTML. We will insert the style definition after the <Document> tag at the beginning of any KML file that will use this style. We cannot use the XMLMap to insert the style definitions at this time since we can only write one table in an XMLMap. This can be accomplished after the KML file is created either algorithmically or manually (not discussed here).

```
<Style id="BA">
    <LineStyle color="ffffffff" width="1" />
    <IconStyle color="ffffffff" scale=".8" >
        <Icon href="http://buffy.hsrc.unc.edu/~martell/bikeafter.png" />
    </IconStyle>
</Style>
```

SUMMARIZATION

The traffic counter data contains tens of thousands of vehicles. We collapse the data into a table having one record per hour and date for each traffic counter, the most granular level we might need. We create indicator variables for daytime, rushhour, weekend, and direction of travel. We create the variables "bikesbefore", "bikesafter", "light", "medium" and "heavy", representing counts of vehicles during the hour. For records in the before study group, bicycle counts feed the variable "bikesbefore" and the "bikesafter" variable is missing; and vice versa.

```
PROC SQL;
…CASE WHEN CALCULATED group='Before' THEN SUM(bike=1)
    ELSE .
    END AS bikesbefore,…
```

**To use XML Mapper, we need to further consolidate to one record per intersection. For our purposes, since there are indicator variables such as daytime and weekend, we accomplish this by using the SUMMARY procedure , creating a table that summarizes across the parameters. We can then later use the "_TYPE_" variable to select a specific set of intersection records to use with a map created by XML Mapper. For CLASS variables we use the intersection, direction of travel, and several time indicators. We bring forward some verbal descriptors in the ID statement. Our analysis variables for summation are counts of the various vehicle types and the recorded hours before and after. We filter the "_TYPE_" value to delete records having a missing value for intersection.

```
PROC SUMMARY DATA=out.studycollapsed SUM CHARTYPE;
CLASS isect dir month weekend daytime rushhour;
ID site corridor cross_street;
VAR bikesbefore bikesafter light med heavy hrsbefore hrsafter;
OUTPUT OUT=out.sumout(WHERE=(_TYPE_ GE '100000'))
SUM (bikesbefore bikesafter light med heavy hrsbefore hrsafter)=
     bikesbefore bikesafter light medium heavy hrsbefore hrsafter;
```

The intersection coordinates are joined with the SUMMARY output data, and traffic rates are calculated from the vehicle counts and hours recorded. For a given intersection, five measures are now available for display: bicycle and light, medium and heavy vehicle rates for the before study group, and bicycle rates for the after study group.

```
PROC SQL;
CREATE TABLE out.sumoutrate AS SELECT A.*, 24*bikesbefore/hrsbefore AS bikeperDayB,
…
b.latitude AS lat, b.longitude AS long FROM out.sumout a, out.addresses b
WHERE a.site=b.site
ORDER BY latitude, longitude;
```

We provide slightly differing geographic coordinates for each vehicle type to avoid having them display atop one another. We choose the offset factors by nudging two placemarks around in Google Earth and noting the resulting coordinate differences by comparing the two placemark properties. In this example, we are nudging the longitudinal coordinate. The traffic rate for the vehicle type provides the third value in the coordinates, the altitude.

```
bikeBloc=COMPRESS(PUT(long,10.6)||','||PUT(lat,10.6)||','||PUT(bikeperdayB,5.1));
bikeAloc=COMPRESS(PUT(long-.00001,10.6)||','||PUT(lat,10.6)||','||
     PUT(bikeperdayA,5.1));
```

We also assign the style designed for each vehicle type and an intersection-level description is constructed. (Surrounding the string with <![CDATA........]]>, required for non-escaped text in KML, is not necessary, as this masking is performed by the SAS engine.)

```
bikeBstyleUrl="#BB";
bikeAstyleUrl="#AB";
…
description='Hours: '||PUT(max(hrsbefore,0),COMMA6.)||' before, '||
     PUT(MAX(hrsafter,0),COMMA6.)||' after<br> '||
     'Bikes: '||put(max(bikesbefore,0),comma4.)||' before, '||
     PUT(MAX(bikesafter,0),COMMA4.)||' after<br> '||'Bikes per day: '||
     … PUT(MAX(heavyperday,0),COMMA5.)||' per day.';
```
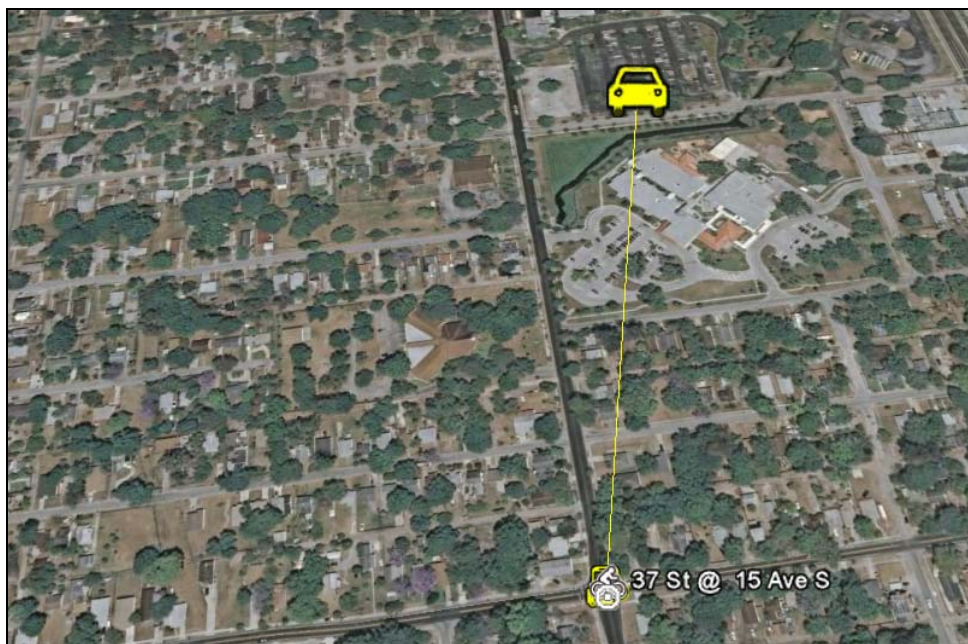

Figure 17 Elevations not within range of one another

Viewing our map, we find an unwelcome surprise. **We find such differences in traffic counts between light (passenger) vehicles and all other vehicle types that when the placemarks are displayed together, the vast elevation differences create a problem. When the view is zoomed in enough to see the differences between the other vehicle types, the light vehicle icon simply does not show up. When zoomed out enough to view the light vehicle icon, the other vehicles type icons are as indistinguishable as a pile of ants. This relative scale problem is shown in Figure 17. We solve this problem by calculating light vehicle rates per hour rather than per day, bringing the elevation numbers more closely in range with the other elevation values. To remind the viewer that the icon represents vehicles/hour rather than vehicles/day, the extruded line weight in the light vehicle style is heavier than in the other styles.

To enhance Google Earth navigation, we populate a set of variables to assign a default viewing perspective, or vantage point. These variables provide name/value pairs for the <LookAt> KML tag structure. We determine these values in Google Earth by finding a view that we like and noting the heading, range and tilt for the view. We derive the coordinate offsets by comparing the placemark coordinates with the view coordinates. This information is visible in Google Earth via tabs in the placemark properties.

```
<LookAt longitude="-82.693449" latitude="27.725906" range="3300" tilt="65"
        heading="45" />

latitude=PUT(lat-.0005,10.6);
longitude=PUT(long,10.6); range=100; tilt=45; heading=30;
```

At long last, we have every variable we could possibly need in a table named "out.tweakedloc"! Subsetting this table, we will create a table to feed each XMLMap by lifting out the records we want and populating our target variables. This process calls for exacting conformity. The KML is written by the XML92 LIBNAME engine. We will use a SET statement in a DATA step to copy a SAS table to the KML library. The following rules apply to the input SAS table we will use in the SET statement. **Drop any extraneous variables not appearing in the XMLMap because variables existing in the SAS table not defined in the map cause SAS to crash. Variables defined in the map but not supplied in the SAS table will simply not exist in the KML output. The input table name can be anything you want. **The output table name must match the name found in the XMLMap. **Input variables must match the XMLMap in name and type. **Character variable lengths cannot exceed the lengths specified in the map; SAS will crash if they are too long.

Here we prepare a SAS table for input to the intersection map. The variable "description" provides the balloon text. **Adding "<table>" to the balloon value removes the line "Directions: To here – From here" that automatically appears in Google Earth balloons.

```
DATA isect(DROP=cross_street lat long corridor);
LENGTH description $ 287 name $ 30;
SET out.tweakedloc(KEEP=cross_street description coordinates long lat corridor
                WHERE=(_type_='100000'));
coordinates=COMPRESS(coordinates); styleUrl='#I';
longitude=PUT(long,10.6); range=100; tilt=45; heading=30;
latitude=PUT(lat-.0005,10.6);
name=TRIM(corridor)||' @ '||TRIM(cross_street);
description=TRIM(description)||'<table>';
RUN;
```

We have the exact set of nine variables defined in "intersection.map". The output KML file must be named "Intersections.kml", the table we write must be named Placemark, and **our XML92 libref name must match the output KML fileref name. Issuing the following statements will create a KML file with a placemark for every intersection with a balloon showing statistics for all traffic recorded at the intersection. Figure 18 shows two panes – the intersections viewed from above, and a balloon that displays if the user clicks an intersection icon.

```
FILENAME is 'my filepath\Intersections.kml';
FILENAME mp 'my filepath\intersection.map';
LIBNAME is XML92 XMLTYPE=XMLMAP XMLMAP=mp;
DATA is.Placemark;  SET isect; RUN;
```

Figure 18 Two views of "Intersections.kml", which provides intersection labels and balloon text

We build a table for each vehicle type, creating and populating the variables specified in the appropriate XMLMap to complete the process of creating the vehicle icons. Opening the resulting files, we can see our intersections with data points from above as seen in Figure 19. Zooming in on an intersection, we see all the vehicle types represented as shown in Figure 20.
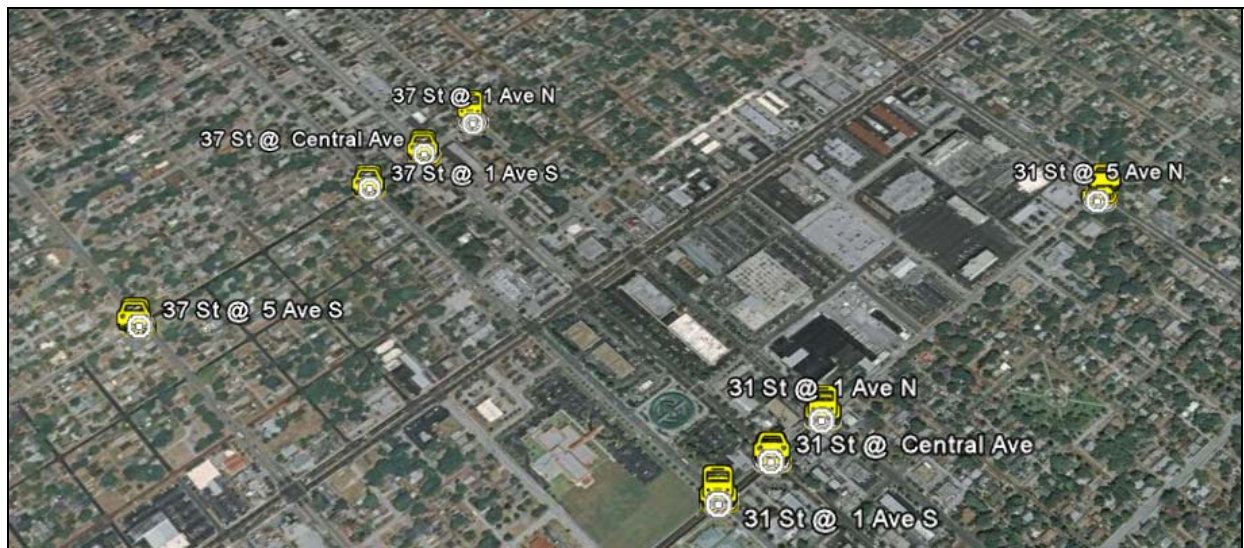

Figure 19 Fully populated intersections viewed from above

Figure 20 Zooming in on a single intersection

Next, we create lines to help visualize traffic volume differences between intersections. The input table for this will consist of a single record per corridor, rather than per intersection. All the intersections along the corridor are represented as pairs of points strung together in the "coordinates" variable. We demonstrate the use of a macro to create a separate output file for each corridor and vehicle type. The two generated lines are shown in Figure 21.

```
%MACRO oneline(corridor,veh,kmlfile,cor,style,extr);
PROC SQL NOPRINT;
CREATE TABLE corridor AS
SELECT DISTINCT name, &veh.rateloc, "&style" AS styleUrl,
  PUT(MEAN(long)-.01,10.6) AS longitude,
  PUT(MEAN(lat)-.04,10.6) AS latitude
FROM out.tweakedloc(WHERE=(_type_='100000' AND corridor="&corridor"))
ORDER BY lat, long;

ALTER TABLE corridor MODIFY name CHAR(20);
SELECT &veh.rateloc INTO :line SEPARATED BY ' ' FROM corridor ;

CREATE TABLE forkml AS
SELECT "&line" AS coordinates, 1 AS tessellate, 'relativeToGround' AS altitudeMode,
  styleUrl, &extr AS extrude, name, longitude, latitude, 3300 AS range, 65 AS tilt,
  45 AS heading
FROM corridor(obs=1);
QUIT;

FILENAME is "…filepath\&kmlfile.&cor..kml";
FILENAME mp "…filepath\&kmlfile..map";
LIBNAME is XML92 XMLTYPE=XMLMAP XMLMAP=mp;

DATA is.Placemark;
 SET forkml;
RUN;
%MEND oneline;

%oneline(37 St,BikeB,BikeBeforeLine,37,#BB,0);
%oneline(37 St,BikeA,BikeAfterLine,37,#BA,0);
```

Figure 21 Lines representing bicycle volumes

USING htmSQL TO GENERATE KML

We will briefly illustrate how htmSQL is really your best friend to generate KML. We assume that the SAS data to be displayed has one record per intersection, as above. We assume the reader has familiarity with htmSQL.

Here we see a stylized placemark KML defining a point for display. This is our template.

```
<Placemark name="" styleUrl="#LB">
   <description> Pinellas Point Dr S
   </description>
   <Point extrude="1" coordinates="-82.675145,27.708944,295.3"
          altitudeMode="relativeToGround" />
   <LookAt longitude="-82.675125" latitude="27.708444"
           range="100" tilt="45" heading="30" />
</Placemark>
```

Assume our SAS table has the variables "placemarkname","description","styleurl" and "coordinates". Here we see the snippet of code to appear in the {eachrow} section of a query.

```
<Placemark name="" styleUrl="{&styleurl}">
  <description>{&description}
  </description>
  <Point extrude="1" coordinates="{&coordinates}"
         altitudeMode="relativeToGround"/>
  <LookAt longitude="{&long}" latitude="{&lat}"
          range="100" tilt="45" heading="30" />
</Placemark>
```

Variable names and types are not restricted. Tags that will contain constant values can simply be hard-coded into the .hsql file, as with "range", "tilt" and "heading" in the example. The limitation of one table per output KML file does not apply. The htmSQL code to create a KML document of some complexity might look like the following.

```
<?xml version="1.0" encoding="UTF-8" ?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<Style id="BB">
    <LineStyle color="ff00ffff" width="2" />
    <IconStyle color="ff00ffff" scale=".8" >
       <Icon href="http://mywebserver/bikebefore.png" />
    </IconStyle>
</Style>
<Style
…other style definitions
</Style>
{QUERY SERVER="myshareserver"}

{SQL}
    SELECT cross_street, bikeBloc, long, lat
    FROM mylib.mysastable
    WHERE _type_='100000'
{/SQL}
```

```
 <Folder id="BikesBefore" name="Bicycle volumes before installation of bikelanes">
{EACHROW}
   <Placemark name="" styleUrl="#BB">
     <description>{&cross_street}</description>
     <Point extrude="1" coordinates="{&bikeBloc}"
            altitudeMode="relativeToGround" />
     <LookAt longitude="{&long}" latitude="{&lat}"
             range="100" tilt="45" heading="30" />
   </Placemark>
{/EACHROW}
 </Folder>
{NOROWS}
  my error message
{/NOROWS}

…REPEAT OF {SQL} THROUGH {/NOROWS} TO CREATE AFTER BIKELANE INSTALLATION FOLDER…

{/QUERY}
</Document>
</kml>
```

The KML file can be generated from a command line. Assuming the htmSQL code is saved as "myplacemarks.hsql" and the desired KML filename is "hereitis.kml":

```
myprompt% /correctpath/cgi-bin/htmSQL myplacemarks.hsql > hereitis.kml
```

The htmSQL.cfg file would need have the following line to prevent the insertion of a Content-type header line.

```
CONTENT-TYPE =
```

We open "hereitis.kml" in Google Earth and see the results in Figure 22. We see two folders in the sidebar navigation area; one is expanded to display the placemarks contained within the folder. There is no need to modify the KML file generated using htmSQL. To create this same file using XMLMapper, we would need to create a separate file for each set of placemarks, assemble them, insert folder tags, and insert style definitions.

The looser SAS input table requirements - no variable name, type, and length restrictions - paired with the ability to construct complex markup from multiple tables make SAS/IntrNet htmSQL a good choice for creating KML files.
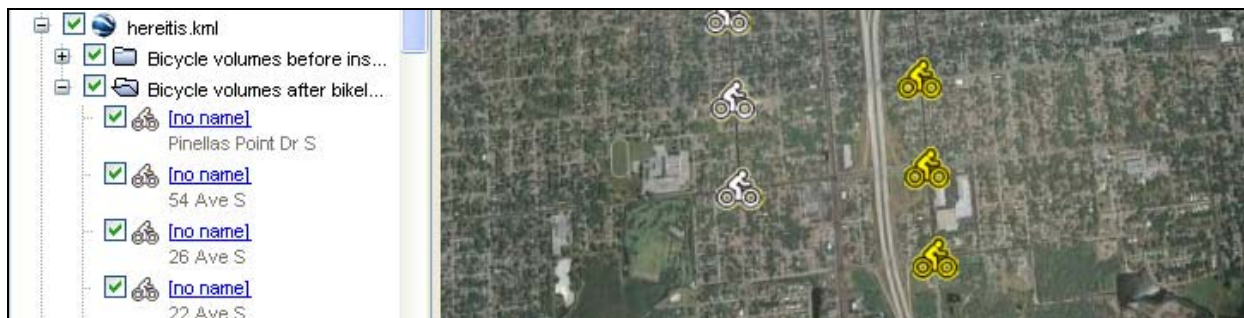

Figure 22 Structured KML file created using htmSQL

CONCLUSION

Meaningful display of statistical data in Google Earth is possible! We have choices in methodology. We can use PUT statements in a DATA step to write data values nested within KML markup; we can create an XMLMap and write KML using the XML92 LIBNAME engine; or we can query data in htmSQL and nest the query results in KML markup.

While most definitely a roller coaster ride, the results are well worth the effort. The reader can avoid derailments by heeding the tips marked with ** throughout the paper. Building the XMLMap by creating and reading a template of the desired result is not difficult. The key to success is preparing a SAS table that conforms to the exact requirements of the XMLMap. Creating KML files using htmSQL is a simpler, more direct process available to those users having the required SAS/SHARE server and web server.

CONTACT INFORMATION

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

Carol Martell
UNC Highway Safety Research Center
730 Martin Luther King, Jr Blvd
Chapel Hill, NC 27514
E-mail: carol_martell@unc.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.