

Distances: Let SAS® Do the Heavy Lifting

Jason O'Day MBA, US Bank

ABSTRACT

SAS® has a very efficient and powerful way to get distances between an event and a customer. Using the tables and code located at <http://support.sas.com/rnd/datavisualization/maponline/html/geocode.html#street> ⁽¹⁾, you can load latitude and longitude to addresses that you have for your events and customers. Once you have the tables downloaded from SAS, and you have run the code to get them into SAS data sets, this paper helps guide you through the rest using PROC GEOCODE and the GEODIST function. This can help you determine to whom to market an event. And, you can see how far a client is from one of your facilities.

INTRODUCTION

If you already have latitude and longitude in your data bully for you, but for the rest of us using data where they are missing the website mentioned in the abstract (<http://support.sas.com/rnd/datavisualization/maponline/html/geocode.html#street>) is a great tool to get them. On that site there are hyperlinks to zipped folders to download that have the csv files needed as well as the SAS code to load them into SAS datasets. I will briefly touch on the files and code for loading these, but will spend the majority of the paper sharing code to map the coordinates and then calculate the differences in locations. To attempt to make this useful to the broader audience I will start with the background of what was the request by the business users and then give the sample code that was used to achieve the results requested.

I will be reviewing the STREET METHOD in this paper but there are other ways of getting distances and you can see some of those in the Recommended Reading portion at the end of the paper.

This is not a new process or even revolutionary, but as with all things that we learn to do as programmers, it started with an existing process that I was taking over. We all know that when we get something new we have to ask good questions on how to do something or how to do it better. That is where this story begins.

PROBLEM CONTEXT

There was a standing process that had been used in the past but the analysts that were running it wanted the SAS Developers to take it over and to update it for other portions of the business. After meeting with them I found that they were getting latitude and longitude from some 'mysterious file' and the distances were being calculated using a formula that was found online by the team. In taking this over I wanted to rewrite most of the process and look for ways to improve on what has been done. I also wanted to get to the bottom of this file that was used to obtain the coordinates. Below are the sample codes and the description of how it works in gathering distances from one location to another as the bird flies.

GATHERING CSV FILES AND IMPORTING INTO SAS DATASETS

When talking about having SAS do the heavy lifting this section demonstrates that more than any other part. SAS already has the files and the code to help you in your quest to apply latitude and longitude to your data. So, in the case of the mysterious file was solved and now we could update the datasets as new files were loaded to SAS Support.

Please see the screen shot below of the web page <http://support.sas.com/rnd/datavisualization/maponline/html/geocode.html#street> (notice this is the third time this site was mentioned and that is not by mistake). For the sake of this paper I selected the most current address information (the one with the arrow pointing at it: StreetLookupData(9.4)-2016.zip). Depending on what version of SAS you have you may have to select a different file. Downloading the zip file can take a long time as the files within are large.

Page Contents

[PROC GEOCODE Overview](#)
[Street Geocoding](#)
[City Geocoding](#)
[Postal Code Geocoding](#)
[ZIP+4 Geocoding](#)
[IP Address Geocoding](#)
[SGF Geocoding Paper](#)

PROC GEOCODE

Geocoding is the conversion of an address into a location. Various parts of an address can be used depending on how much precision is wanted in that location.

PROC GEOCODE is a SAS/GRAPH procedure for geocoding by street address, city, ZIP code, ZIP+4 and IP address. Each geocoding method requires specific lookup data. Some of the lookup data is installed with SAS, some is available below, and others downloaded from government sources or data vendors. See the SAS/GRAPH documentation for details on using these lookup data sets with the GEOCODE Procedure.

Street Geocoding

Street geocoding for the U.S. was added to PROC GEOCODE in the third maintenance release of SAS 9.2 (9.2M3). U.S. lookup data is generated from Census Bureau [TIGER/Line shapefiles](#).

Canadian street geocoding was added in SAS 9.4. Canadian lookup data is generated from GeoBase [National Road Network](#) (NRN) files. The Canadian data will not work with PROC GEOCODE releases prior to SAS 9.4.

The zipped files below contain prebuilt geocoding data files, a ReadMe.txt file with instructions, and a SAS program to import the CSV data files into data sets. Some of the zipped files are large, up to 1.7 Gb.

In addition to downloading prebuilt U.S. or Canadian data, we also provide the SAS programs where you can create lookup data from the original source. The programs allow you to download TIGER or NRN shapefiles for specific U.S. counties or Canadian provinces and create the lookup data for more limited regions.

The format of the street lookup data sets changed in SAS 9.4. Be careful to download files from the appropriate section below for your SAS release.

- **SAS 9.4 or Later**

Prebuilt U.S. street lookup data for specific TIGER release (created with Ver. 14 of TIGER2Geocode):

[StreetLookupData \(9.4\)-2016.zip](#)
[StreetLookupData \(9.4\)-2015.zip](#)
[StreetLookupData \(9.4\)-2014.zip](#)

The code in the zip folder has instructions for updating it, so that you can put the path for the location of the datasets and the location of the csv files to be loaded:

```
| Summary : 1) Download Census Bureau TIGER/Line files for the desired US
|           counties and states from www.census.gov/geo/www/tiger.
|           2) Unzip all downloaded TIGER files into a single directory.
|           3) Set required TIGERPATH macro var to that location.
|           4) Set required DATASETPATH macro var to directory where the
|              final PROC GEOCODE lookup data sets are to be written.
|           5) Specify desired optional macro vars from above list.
|           6) Submit TIGER2GEOCODE.sas file to compile macro programs.
|           7) Invoke the TIGER2GEOCODE macro program.
|           8) Check log for errors or warnings.
|           9) Review geocoding lookup data sets.
|           10) Test lookup data using PROC GEOCODE street method.
|                See PROC GEOCODE documentation.
|           11) Create backup of PROC GEOCODE lookup data sets.
|           12) If no longer needed, delete temporary files and data sets:
|                a) Downloaded TIGER/Line zip files
|                b) Unzipped TIGER files in TIGERPATH
|                c) Individual county data sets in WORK
|                d) Interim files and data sets in WORK
|           12) Lookup data sets are ready for street level geocoding.
|                See LOOKUPSTREET= option in PROC GEOCODE doc for instructions.
|
| Example : %let TIGERPATH=C:\Geocoding\TIGER files;    /* Directory with unzipped shapefiles
|              %let DATASETPATH=C:\Geocoding\Lookup data; /* Output data set directory
|              %let DATASETNAME=DELAWARE_;             /* Output data set name prefix
|              %include 'C:\Geocoding\TIGER2GEOCODE.sas'; /* Compile macro program
|              %TIGER2GEOCODE                          /* Invoke TIGER import macro
|
|-----*/
```

Once you make these updates to the code and you have the csv files (TIGER files) in a single directory you can run the code. In the case of this paper we have saved the SAS dataset, USM.sas7bdat, to a library named SASDATA.

PROC GEOCODE TO ASSIGN LATITUDE AND LONGITUDE

Once the code has completed running above and the dataset USM is created in a permanent location (sasdata in this case), we can start to assign the latitude and longitude to the observations in the data. In

the code below you can see that we need to assign the METHOD and the variables that follow in the PROC GEOCODE process.

The fields: ADR_1, STATE, CTY and ZIP_CD are from the dataset MemberAddress1, which is the input table that we need to match with the USM table in order to get the latitude and longitude variables.

```
PROC GEOCODE DATA=WORK.MemberAddress1 /* Input table that needs Coordinates table */
  OUT=WORK.mbrs_geo /* Output table that needs Coordinates table */
  METHOD=street /* METHOD Type (street in our case) */
  LOOKUPSTREET=sasdata.usm /* SAS Address lookup table */
  ADDRESSVAR=adr_1 /* Address variable in input data set */
  ADDRESSSTATEVAR=state /* State variable in input data set */
  ADDRESSCITYVAR=cty /* City variable in input data set */
  ADDRESSZIPVAR=zip_cd /* Zip Code variable in input data set */
;
RUN;
```

You can see that field 'Y' is latitude and 'X' is longitude. Other fields that we can use are '_MATCHED_' and '_SCORE_'. '_MATCHED_' will tell you what field it was matched on whether it was ADDRESS, ZIP or CITY. '_SCORE_' gives a numeric value of how well it matched from USM to your dataset, the higher the better. We do not use '_SCORE_' to filter in this process but I have seen other instances where developers do filter on it. For more information on these fields and how you can use them see the Recommended Reading section.

```
PROC SQL;
  CREATE TABLE WORK.mbrs_geo1 AS
  SELECT DISTINCT
    y AS lat
    , x AS lon
    , _matched_
    , _score_
    , person_id
    , state
  FROM WORK.mbrs_geo
  ORDER BY person_id
;QUIT;
```

Below is the same code as above but for the data we need to merge later in order to get the distances between the members and the event locations.

```
PROC GEOCODE DATA=WORK.unq_envt_location /* Input table that needs Coordinates table */
  OUT=WORK.event_geo /* Output table that needs Coordinates table */
  METHOD=street /* METHOD Type (street in our case) */
  LOOKUPSTREET=sasdata.usm /* SAS Address lookup table */
  ADDRESSVAR=address /* Address variable in input data set */
  ADDRESSSTATEVAR=state /* State variable in input data set */
  ADDRESSCITYVAR=city /* City variable in input data set */
  ADDRESSZIPVAR=zip_cd /* Zip Code variable in input data set */
;
RUN;

PROC SQL;
  CREATE TABLE WORK.event_geo1 AS
  SELECT DISTINCT
    y AS lat
    , x AS lon
```

```

        ,_matched_
        ,_score_
        ,submarket
        ,address
        ,city
        ,state
        ,zip_cd
FROM WORK.event_geo
WHERE UPCASE(_matched_) ^= 'NONE' /*Keeping only where there is a Match*/
ORDER BY submarket
;QUIT;

```

USING THE GEODIST FUNCTION RATHER THAN CALCULATING THE DISTANCES WITH SINE AND COSINE

Once we have the latitude and longitude we can begin to put the two datasets together and calculate the distances. When I met with the business to go over their code, they had used the calculation below. I could follow it but was looking for a better way. I researched the calculation and found it in a paper by Mike Zdeb, "Driving Distances and Times Using SAS® and Google Maps" and is called the Haversine formula:

$$d3 = 3949.99 * \arcsin(\sin(\text{lat1}) * \sin(\text{lat2}) + \cos(\text{lat1}) * \cos(\text{lat2}) * \cos(\text{long2} - \text{long1})).$$

This is one way to do it but as I researched I found that the function GEODIST can be used to calculate the distance between two points in kilometers, miles, degrees and radians. This seemed like a better way for me as I prefer a universal way to code and if new requests for change come from the business and they want to calculate in kilometers instead of miles there are very small updates to the code that is needed. I prefer to make my life as easy as possible when developing code.

If we look at the structure of the function it is very simple:

GEODIST(latitude-1, longitude-1, latitude-2, longitude-2 <,options>) ⁽³⁾

The options are either 'K' for kilometers, 'M' for miles, 'D' for degrees or 'R' for radians.

In the code below we are joining the tables and calculating the distances in miles and are only keeping the data where the distance is less than 10 miles and it is not null.

```

PROC SQL;
CREATE TABLE WORK.mbr_evnt_geocoding AS
SELECT
    A.*
    ,E.submarket
    ,E.lat           AS event_lat
    ,E.lon           AS event_lon
    ,E.state
    ,GEODIST(A.lat, A.lon, E.lat, E.lon, 'M') AS distancetoevent
    /* M is used for miles in our case */
FROM WORK.mbrs_geol AS a
LEFT JOIN WORK.event_geol AS e ON A.state = E.state
WHERE CALCULATED distancetoevent <=10
AND CALCULATED distancetoevent ^=.
ORDER BY A.person_id
;QUIT;

```

FILTER THE DATA BASED ON THE MAXIMUM DISTANCE FOR MARKETING PURPOSES

Finally, the business requested that we develop the code to flag the members that were within a certain distance to specific submarkets for the events. In the code below we took those parameters and applied them to the submarkets: 'MINNEAPOLIS', 'ST. PAUL', and 'RICHFIELD'. We cleaned up the data by removing duplicates and created a finished dataset to send to the marketing team. They could then put together the mailing lists for the campaigns.

```
DATA WORK.mbr_evnt_geocoding_2;
  SET WORK.mbr_evnt_geocoding;
  IF UPCASE(submarket) = 'MINNEAPOLIS'
    AND DistanceToEvent <= 5.5 THEN evnt_within = 'Y';
  ELSE IF UPCASE(submarket) = 'ST. PAUL'
    AND DistanceToEvent <= 6.0 THEN evnt_within = 'Y';
  ELSE IF UPCASE(submarket) = 'RICHFIELD'
    AND DistanceToEvent <= 7.5 THEN evnt_within = 'Y';
  ELSE evnt_within = 'N';

RUN;

PROC SORT DATA= WORK.mbr_evnt_geocoding_2;
  BY distancetoevent;

RUN;

PROC SORT DATA= WORK.mbr_evnt_geocoding_2
  OUT=WORK.mbr_evnt_geocoding_fnl NODUPKEY;
  BY person_id;

RUN;

PROC SQL;
  CREATE TABLE output.radius_check AS
  SELECT DISTINCT
    A.peron_id
    ,CASE WHEN C.evnt_within = 'Y' THEN 'Y' ELSE 'N' END AS event_ind
  FROM WORK.mbrs_geol AS a
  LEFT JOIN WORK.mbr_evnt_geocoding_fnl AS c ON A.ucps_id = C.ucps_id
;QUIT;
```

CONCLUSION

In summary, this paper can really be broken down to a few components that will make your life easier as far as gathering the coordinates for locations and then calculating those differences using SAS. It is made easy because SAS first gives you the latitude and longitude files as well as the code to load them to your environment. Second, SAS gives you the power of PROC GEOCODE which we have barely scratched the surface on in this paper. If you want more information on this procedure I suggest reading a few of the resources in the Recommended Reading portion below. Finally, SAS gives the ease of calculating the distances using the GEODIST function. You can try gathering the coordinates yourself and even using complicated calculations to get the distances, but wouldn't you prefer to have SAS do the heavy lifting?

REFERENCES

1. "SAS Maps Online." SAS Support. 2017. <http://support.sas.com/rnd/datavisualization/mapsonline/html/geocode.html#street>
2. Zdeb, Mike. "Driving Distances and Times Using SAS® and Google Maps" SAS Global Forum 2010. <http://support.sas.com/resources/papers/proceedings10/050-2010.pdf>

3. "SAS Support Documentation"
<http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a003113162.htm>

ACKNOWLEDGMENTS

Thank you to Kirk Paul Lafler for proof reading my paper as well as my presentation. That is a huge help for a first time presenter and I really appreciate it.

RECOMMENDED READING

- Massengill, Darrell and Odom, Ed. 2013 "PROC GEOCODE: Finding Locations Outside the U.S." SAS Global Forum 2013. <http://support.sas.com/rnd/papers/sasgf13/Geocode2013.pdf>
- Massengill, Darrell and Odom, Ed. 2013 "PROC GEOCODE: Now with Street-Level Geocoding." SAS Global Forum 2010. <http://support.sas.com/resources/papers/proceedings10/332-2010.pdf>

CONTACT INFORMATION

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

Jason O'Day MBA
Jason.oday@usbank.com