Paper 252-2008

Put Your Customers on the Map: Integrating SAS/GRAPH® and Google Earth

Daniël Kuiper, SAS Institute, the Netherlands
Koen Vyverman, SAS Institute, the Netherlands

ABSTRACT

Ever fancied creating a Google Earth map from SAS® data? Or, conversely, loading Google Earth placemarks into a SAS data set? We propose an XML-based interface between SAS and Google Earth, allowing easy exchange of data between the two applications. We show how a folder of Google Earth placemarks can be plotted on a SAS/GRAPH® map by means of an XML Map created with SAS® XML Mapper. Going the other way, we transform a SAS/GRAPH customer distribution plot into an interactive Google Earth satellite view.

INTRODUCTION

In the fall of 2006, the local Technical Support team of SAS Institute in the Netherlands manned an information booth at the annual SAS Forum Netherlands. In order to attract the attention of passers-by, we came up with the idea to project the Google Earth view of the Netherlands on a large screen, and we invited visitors to define a placemark locating their offices, thereby effectively creating a visual map of our Dutch SAS customers.

Although we noticed at the time that many people paused in front of the booth to stare at the screen, we didn't think much of it until some time after the event, when Technical Support began to receive phone calls from customers asking how to achieve the integration between SAS and Google Earth that they had seen at the Forum. Considering the situation, we felt obliged to come up with a mechanism to achieve this perceived integration.

In this paper we make a round-trip from Google Earth (the free version) to SAS and back again. For the first leg of the trip, we export a group of Google Earth placemarks as an XML file. By means of the SAS XML Mapper, we create an XML Map that enables us to read the exported geographical data into SAS and use the data to generate a SAS/GRAPH plot. For the return trip, we start with a SAS/GRAPH plot and turn its data into an XML file readable by Google Earth. We use SAS 9.2, which has new functionality for the creation of XML output via XML Maps.

FIRST LEG: FROM GOOGLE EARTH TO SAS

EXPORT PLACEMARKS AS XML

To illustrate the process step by step, we create a folder of placemarks in Google Earth and name it `Flatland Faves`. In this folder we store placemarks for a number of our favorite places in the Netherlands. Right-clicking on the folder name (Figure 1) and selecting **`Save As`** brings up a dialog box enabling export as either a KMZ file or a KML file. We choose the KML format, and save `Flatland Faves.kml` to our project directory `C:\ge2sas2ge\xml input\`.

KML, or Keyhole Markup Language, is Google Earth's way of storing all types of attributes—such as coordinates, shape and color of markers, label text, and opacity—of placemarks and of groups or folders of placemarks. KMZ is a compressed form of KML, which we don't use in this process. For a full description of the KML language, see the Google "KML Documentation Introduction" (Google, 2007), especially the "KML Reference," which describes the syntax and usage of all the KML tags.

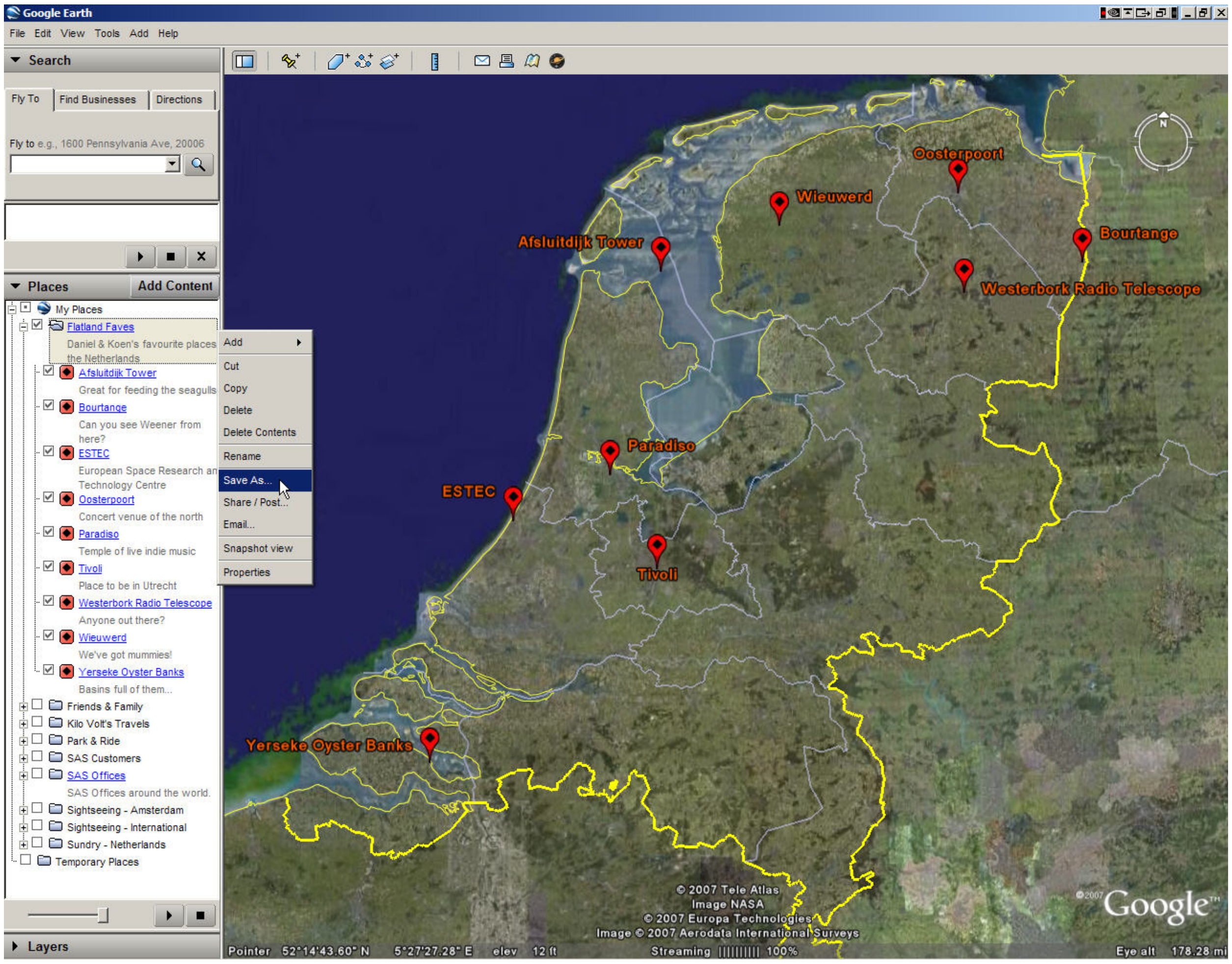


Figure 1. Saving a Placemarks Folder as a KML File

It is beyond the scope of this paper to discuss the KML specification in any depth, since we'll merely be using some very basic elements of it. For the time being, suffice it to say that KML looks a lot like XML, as can be seen in the following excerpts from our `Flatland Faves.kml` file:

```
<?xml version="1.0" encoding="UTF-8"?>❶
<kml xmlns="http://earth.google.com/kml/2.2">❷
<Document>
  <name>Flatland Faves.kml</name>
  <Style id="sh_ylw-pushpin_copy6">❸
    <IconStyle>
      <color>ff0000ff</color>
      <scale>1.3</scale>
      <Icon>
        <href>http://maps.google.com/mapfiles/kml/paddle/red-diamond.png</href>❹
      </Icon>
      <hotSpot x="32" y="1" xunits="pixels" yunits="pixels"/>
    </IconStyle>
    ...
    ...
  </Style>
```

KML is in fact based on the XML standard, and it is important to know that the tag names are case sensitive and that tags within certain nested elements must appear in a specific order. At the top of the file we observe the following elements:

- ❶ A mandatory XML header specifying the XML syntax version and a character encoding.
- ❷ An XML namespace declaration.
- ❸ A number of style definitions used to control the look and feel of the Google Earth placemarks, such as the type of marker and its size and color.
- ❹ Possible dynamic content. We won't use this within the context of this paper, but some fairly cool applications have been built that rely upon this feature (Mintz, 2008).

Scrolling further down, the placemark for one of our favorite places shows up like this:

```
<Placemark>
  <name>Paradiso</name>
  <description>Temple of live indie music</description>
  <LookAt>
    <longitude>4.883922925259545</longitude>
    <latitude>52.36208499547839</latitude>
    <altitude>0</altitude>
    <range>139.6656827351202</range>
    <tilt>1.200079114380297e-010</tilt>
    <heading>-0.80010711659419</heading>
    <altitudeMode>relativeToGround</altitudeMode>
  </LookAt>
  <styleUrl>#msn_ylw-pushpin_copy6</styleUrl>
  <Point>
    <coordinates>4.883844702895061,52.36214030828128,0</coordinates>
  </Point>
</Placemark>
```

This is where things become interesting. The `name` tag contains the label of the placemark, the `description` tag contains the text that appears in the placemark's balloon, and the `Point` tag lists the geographical coordinates of the placemark: longitude, latitude, and optionally altitude. These are the items we want to extract from the KML file that is generated from Google Earth and export to SAS.

Furthermore, though of no immediate concern to us, the `LookAt` tag contains the default viewing information for the current placemark. In essence, it defines how one sees the geographic location when Google Earth flies to it when the placemark is double-clicked. The `styleUrl` tag refers to one of the presentation styles defined earlier on.

In order to extract the placemark attributes that are of interest to us, we use the SAS XML Mapper application. For the XML Mapper to recognize our `Flatland Faves.kml` file as XML, we simply rename it as `Flatland Faves.xml`

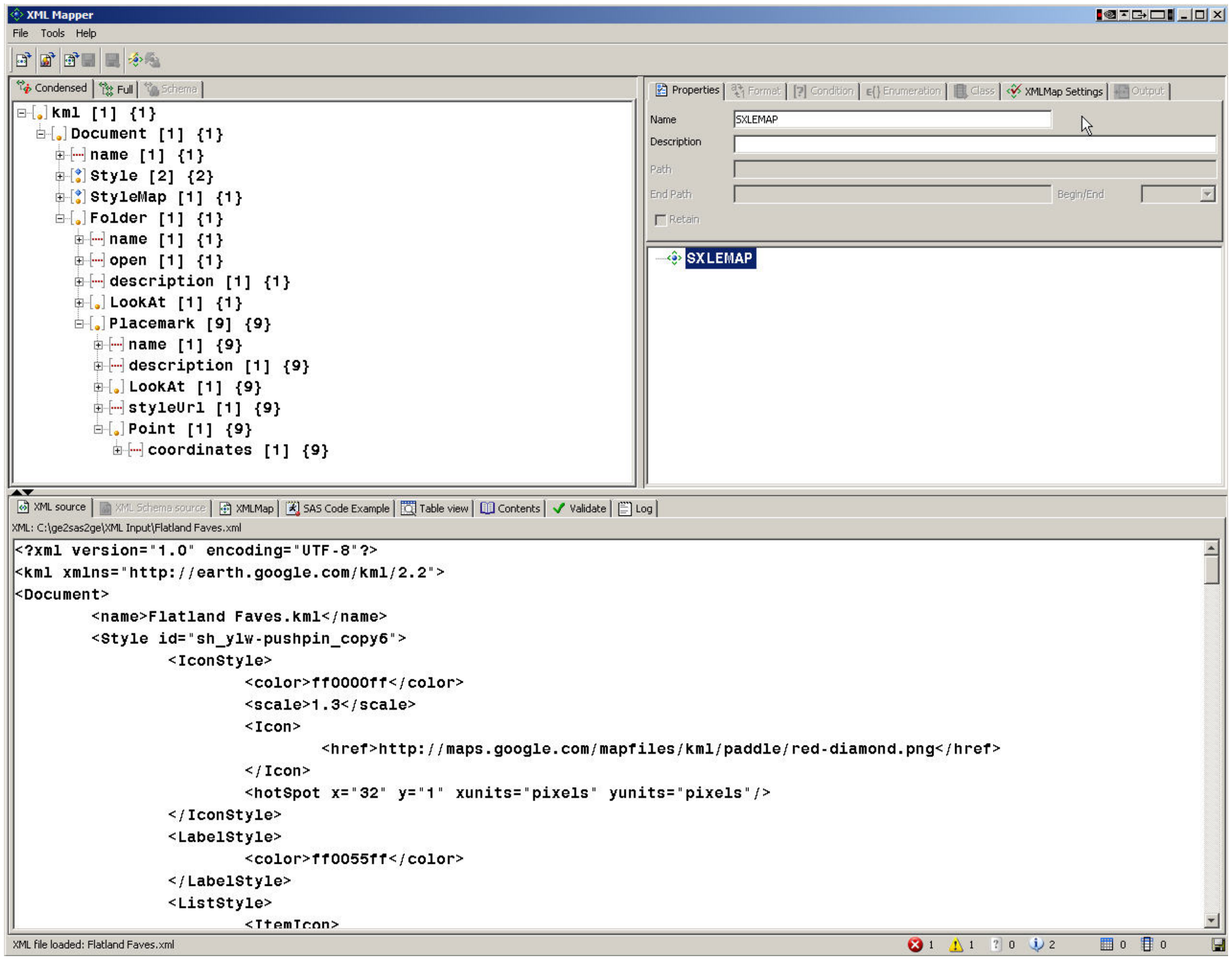
USE SAS XML MAPPER TO CREATE AN IMPORT XML MAP

To facilitate transferring data from XML generated by a third-party application into a SAS data set, the XML LIBNAME engine supports a number of markup types, such as CDISC (Clinical Data Interchange Standards Consortium), MS Access, and Oracle. These markup types can be specified via the XMLTYPE option on the LIBNAME statement. The number of supported XML types is limited, though, and in order to import XML that does not conform to any of these predefined markup types, one must create an XML Map.

In this section we use the SAS XML Mapper application to define an XML Map based on the `Flatland Faves.kml` file generated by Google Earth. Defining this map will enable us to read the embedded data in the file correctly and import it into SAS using very few lines of SAS code. The making of a good XML Map is a bit of manual work, but it needs to be done only once. Later, we'll tweak our XML Map so that it can also be used to export SAS data back into the type of XML that Google Earth can read. For now we focus on importing data into SAS.

Essentially, an XML Map is a logical mapping from hierarchical tag-embedded content in an XML file to one or more SAS data sets. This mapping identifies certain XML tags as data set variables and the corresponding tag content as the values to populate the variables with. The SAS XML Mapper makes it easy to create such a map by dragging and dropping tags from a tree-view representation of a given XML file into a data set structure (that is, the map). Once that's done, variable types and lengths as well as formats can be adjusted.

We start the SAS XML Mapper application, and then select **`Open XML`** from the **`File`** menu and navigate to `C:\ge2sas2ge\xml input\` to open our `Flatland Faves.xml` file. We then look for the items we want to extract by expanding the hierarchical structure: **`Document`** ⇨ **`Folder`** ⇨ **`Placemark`** ⇨ **`Point`** (Figure 2).


XML Mapper
File Tools Help
Condensed Full Schema
Properties Format Condition Enumeration Class XMLMap Settings Output
Name
Description
Path
End Path
Begin/End
Retain
XML source XML Schema source XMLMap SAS Code Example Table view Contents Validate Log

```
kml [1] {1}
  Document [1] {1}
    name [1] {1}
    Style [2] {2}
    StyleMap [1] {1}
    Folder [1] {1}
      name [1] {1}
      open [1] {1}
      description [1] {1}
      LookAt [1] {1}
      Placemark [9] {9}
        name [1] {9}
        description [1] {9}
        LookAt [1] {9}
        styleUrl [1] {9}
        Point [1] {9}
          coordinates [1] {9}
```

Name: SXLEMAP

SXLEMAP

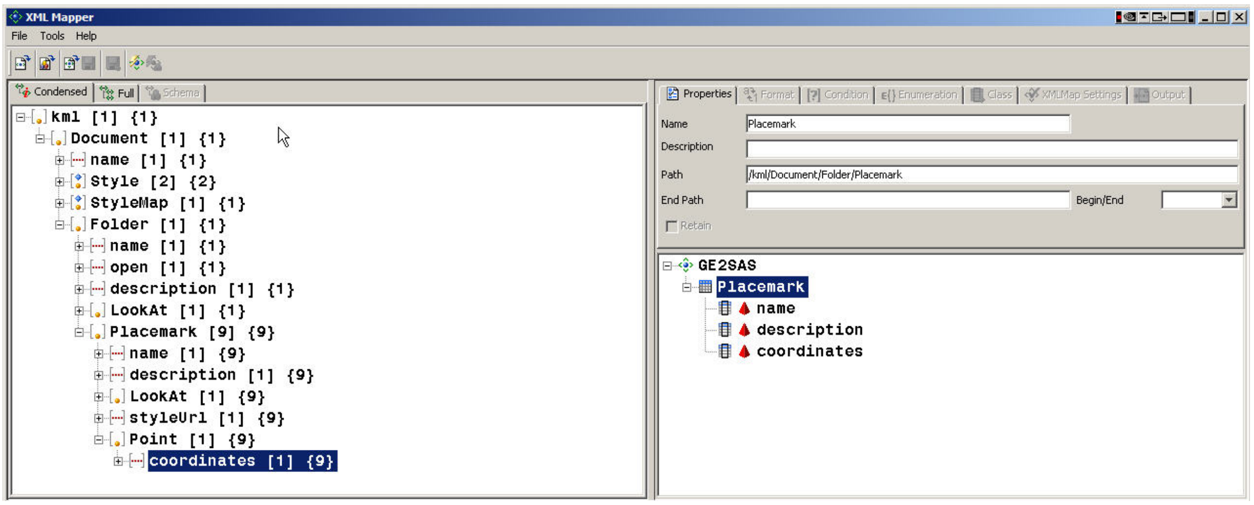
XML: C:\ge2sas2ge\XML Input\Flatland Faves.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
<Document>
	<name>Flatland Faves.kml</name>
	<Style id="sh_ylw-pushpin_copy6">
		<IconStyle>
			<color>ff0000ff</color>
			<scale>1.3</scale>
			<Icon>
				<href>http://maps.google.com/mapfiles/kml/paddle/red-diamond.png</href>
			</Icon>
			<hotSpot x="32" y="1" xunits="pixels" yunits="pixels"/>
		</IconStyle>
		<LabelStyle>
			<color>ff0055ff</color>
		</LabelStyle>
		<ListStyle>
			<ItemIcon>
```

XML file loaded: Flatland Faves.xml 1 1 0 2 0 0

Figure 2. Exploring a Google Earth KML File in SAS XML Mapper

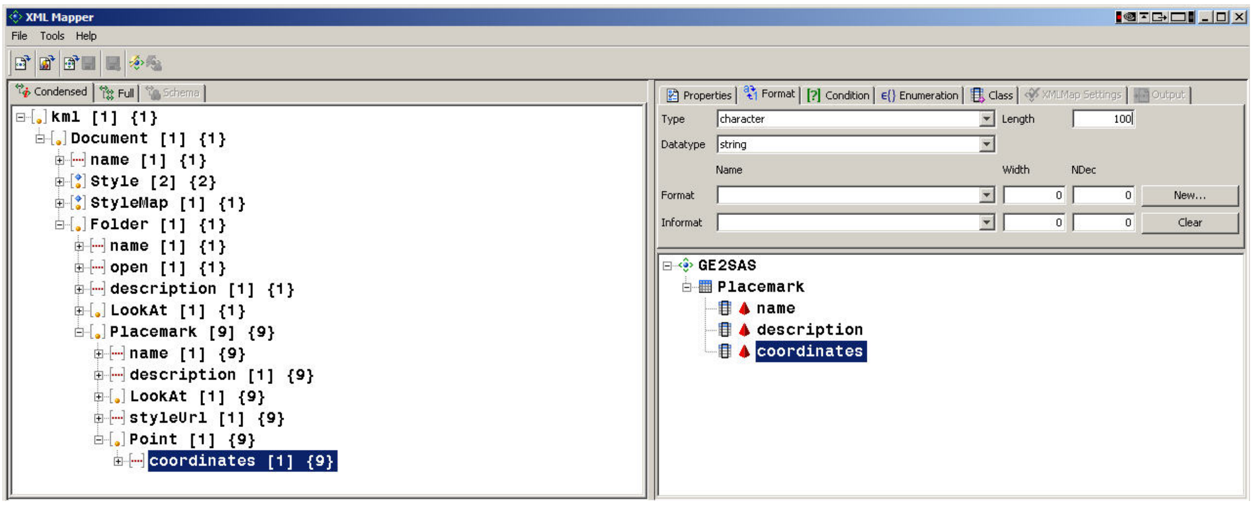
To create an XML Map, we change the default name `SXLEMAP` to `GE2SAS`. We are interested in getting placemark data into SAS, and we want to create a SAS data set with a record for each placemark in the KML source. To define a table named `Placemark`, we drag the `Placemark` item from the left pane into the Map pane on the right. We want some columns in this `Placemark` table, so we drag the placemark attributes `name`, `description`, and `coordinates` onto the `Placemark` table to define the columns `name`, `description`, and `coordinates` (Figure 3).



Name: Placemark
Description:
Path: /kml/Document/Folder/Placemark
End Path:

Figure 3. Defining a Table Based on XML Tags

Looking at the **Format** tab, we see that the XML Mapper application has picked values for the length of our three character variables, based on the data available in the `Flatland Faves.xml` file. We decide we need some more slack, however, because we want to be able to use our XML Map for any set of Google Earth placemarks. So we change the length of the `name` column to 50. We bump the lengths for `description` and `coordinates` to 100 (Figure 4).



Type: character Length: 100
Datatype: string

	Name	Width	NDec
Format		0	0
Informat		0	0

Figure 4. Editing the Length Attribute of the Columns

As we were doing all this clicking around, the XML Mapper has been quietly generating an XML Map for us. We select **Save XML Map As** from the **File** menu and store the XML Map as `C:\ge2sas2ge\xml maps\ge2sas.map`.

An XML Map is just a text file, and `ge2sas.map` contains the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ############################################################ -->
<!-- 2007-12-31T13:45:26 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 9.2.0.000000_v920c_20071205_7927 -->
<!-- ############################################################ -->
<!-- ###  Validation report                                    ### -->
<!-- ############################################################ -->
<!-- XMLMap version (1.2) is an older version. -->
<!-- XMLMap validation completed successfully. -->
<!-- ############################################################ -->
<SXLEMAP name="GE2SAS" version="1.2">❶
```

```
    <!-- ############################################################ -->
    <TABLE name="Placemark">❷
      <TABLE-PATH syntax="XPath">/kml/Document/Folder/Placemark</TABLE-PATH>
      <COLUMN name="name">❷
        <PATH syntax="XPath">/kml/Document/Folder/Placemark/name</PATH>❸
        <TYPE>character</TYPE>
        <DATATYPE>string</DATATYPE>
        <LENGTH>50</LENGTH>
      </COLUMN>
      <COLUMN name="description">❷
        <PATH syntax="XPath">/kml/Document/Folder/Placemark/description</PATH>
        <TYPE>character</TYPE>
        <DATATYPE>string</DATATYPE>
        <LENGTH>100</LENGTH>
      </COLUMN>
      <COLUMN name="coordinates">❷
        <PATH syntax="XPath">/kml/Document/Folder/Placemark/Point/coordinates</PATH>
        <TYPE>character</TYPE>
        <DATATYPE>string</DATATYPE>
        <LENGTH>100</LENGTH>
      </COLUMN>
      </TABLE>
  </SXLEMAP>
```

A few things to note about this XML Map:

❶ SXLEMAP stands for SAS XML LIBNAME Engine Map. The version of the XML Map syntax is 1.2, which is the default setting after installing SAS XML Mapper, but it can be changed to another version in the XML Mapper Options window. The XML Map syntax for version 1.2 is fully documented at the SAS Support site (SAS, 2007). Different versions of the XML Map syntax offer different functionality, and as we shall see further on, version 1.9 is required in order to write XML files via an XML Map.

❷ These are the table and column definitions as we picked them in the XML Mapper.

❸ For each element, the map specifies an Xpath, which is a UNIX-like path notation defining exactly where the element can be found in the nested hierarchies of the XML file.

READ DATA INTO SAS USING THE XML MAP

Now we're ready to read the data into SAS. In the SAS Program Editor, we submit the following code:

```
  filename gedata 'c:\ge2sas2ge\xml input\flatland faves.xml';❶
  filename g2smap 'c:\ge2sas2ge\xml maps\ge2sas.map';

  libname gedata xml xmlmap=g2smap;❷
  libname sasdata 'c:\ge2sas2ge\sas data';

  data sasdata.placemark;❸
    set gedata.placemark;
  run;

  libname gedata clear;❹
  filename gedata clear;
  filename g2smap clear;
```

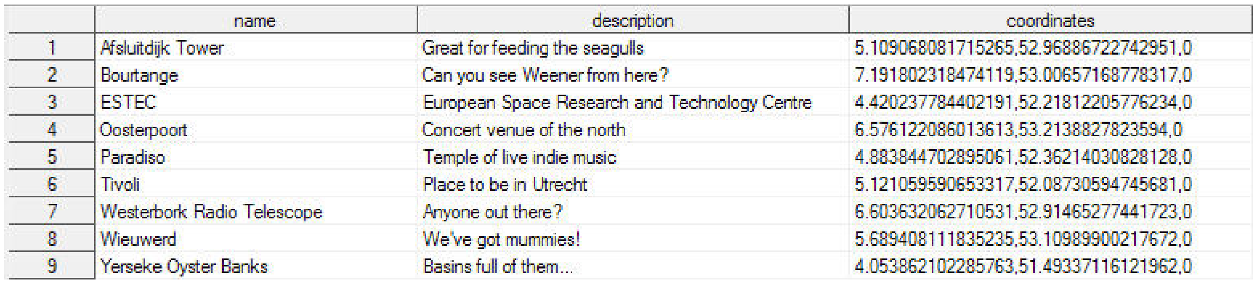
And that's basically it! A few comments:

❶ Define a fileref for the XML file saved from Google Earth, and another one for the XML Map which is needed to interpret the XML content correctly.

❷ Assign a libref with the XML access engine that uses the XML Map, and one to store the extracted SAS data sets.

❸ Extract a SAS data set from the Google Earth XML file. The contents are shown in the following figure (Figure 5).

❹ Clean up; these are no longer needed.



	name	description	coordinates
1	Afsluitdijk Tower	Great for feeding the seagulls	5.109068081715265,52.96886722742951,0
2	Bourtange	Can you see Weener from here?	7.191802318474119,53.00657168778317,0
3	ESTEC	European Space Research and Technology Centre	4.420237784402191,52.21812205776234,0
4	Oosterpoort	Concert venue of the north	6.576122086013613,53.2138827823594,0
5	Paradiso	Temple of live indie music	4.883844702895061,52.36214030828128,0
6	Tivoli	Place to be in Utrecht	5.121059590653317,52.08730594745681,0
7	Westerbork Radio Telescope	Anyone out there?	6.603632062710531,52.91465277441723,0
8	Wieuwerd	We've got mummies!	5.689408111835235,53.10989900217672,0
9	Yerseke Oyster Banks	Basins full of them...	4.053862102285763,51.49337116121962,0

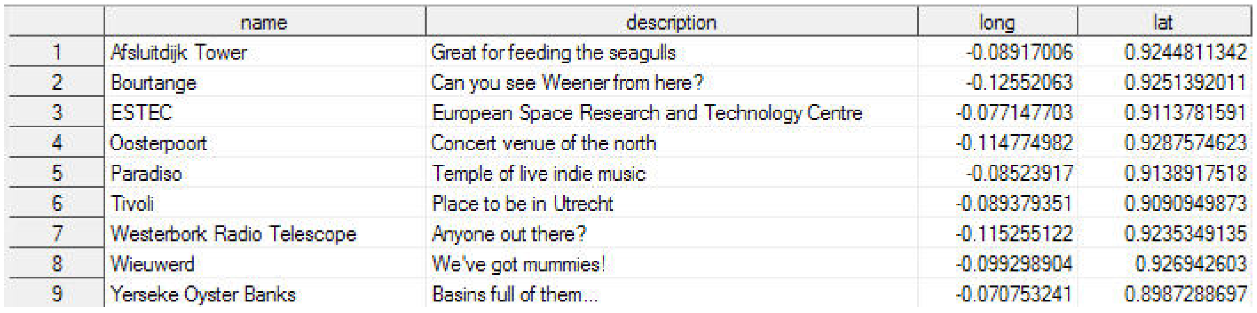
Figure 5. Google Earth Placemark Data in the `sasdata.placemark` Data Set

TRANSFORM DATA AND CREATE AN ANNOTATE DATA SET FOR SAS/GRAPH USE

Once we got to this point, we thought that actually using these Google Earth coordinates in conjunction with SAS/GRAPH maps to make a nice plot in SAS was going to be a piece of cake. As it turns out, the process is not exactly trivial; hence the detailed description in the following pages.

The `coordinates` variable as extracted from the XML contains three comma-separated values for longitude, latitude, and altitude. In the following DATA step, we strip these apart for further use. We ignore the `altitude` and convert `longitude` and `latitude` from degrees and decimal fractions of a degree into radians, because that's how coordinates are stored in the SAS/GRAPH map data sets. We also reverse the sign of `longitude`, because Google Earth and SAS/GRAPH Maps use opposing sign conventions. Note that as of SAS 9.2, the SCAN function supports a fourth argument containing modifiers such as `r`, which removes leading and trailing blanks from the result. The resulting data set, `work.coordinates`, is shown in Figure 6.

```
data coordinates(drop=delimiters modifiers coordinates pi);
  set sasdata.placemark;
  length
    delimiters $ 1
    modifiers  $ 2
    pi
    long
    lat  8
    ;
  pi=constant('pi');
  delimiters=',';
  modifiers='r';
  long=-pi/180*input(scan(coordinates,1,delimiters,modifiers),20.);
  lat=pi/180*input(scan(coordinates,2,delimiters,modifiers),20.);
run;
```



	name	description	long	lat
1	Afsluitdijk Tower	Great for feeding the seagulls	-0.08917006	0.9244811342
2	Bourtange	Can you see Weener from here?	-0.12552063	0.9251392011
3	ESTEC	European Space Research and Technology Centre	-0.077147703	0.9113781591
4	Oosterpoort	Concert venue of the north	-0.114774982	0.9287574623
5	Paradiso	Temple of live indie music	-0.08523917	0.9138917518
6	Tivoli	Place to be in Utrecht	-0.089379351	0.9090949873
7	Westerbork Radio Telescope	Anyone out there?	-0.115255122	0.9235349135
8	Wieuwerd	We've got mummies!	-0.099298904	0.926942603
9	Yerseke Oyster Banks	Basins full of them...	-0.070753241	0.8987288697

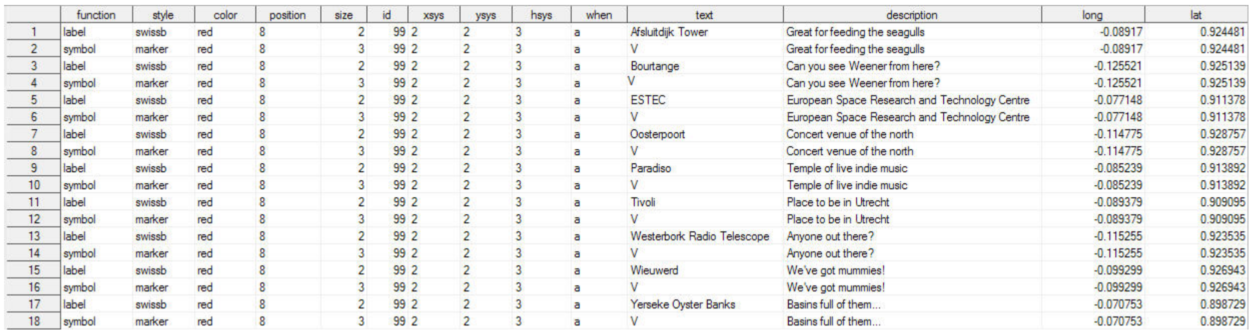
Figure 6. The `work.coordinates` Data Set at This Point

We plan to use PROC GMAP to display a map of the Netherlands (using the map data set `maps.ntlands`), with our extra points added by means of the Annotate facility. For a comprehensive overview of PROC GMAP, PROC GPROJECT, and map coordinates in conjunction with the use of Annotate, see Davis (1997).

The following code turns the `work.coordinates` data set into an Annotate data set. Our `long` and `lat` variables contain what SAS/GRAPH refers to as unprojected coordinates, which is to say they are simple spherical coordinates. Although PROC GMAP will work with unprojected coordinates, realistic pictures are obtained only when projected coordinates are used. And to apply a projection to a large group of coordinates, PROC GPROJECT can be used. However, to obtain consistent results, we need to project our `long` and `lat` values together with the unprojected coordinates in `maps.ntlands`. So we add a bogus map `id` code 99, because PROC GPROJECT requires an `id` variable.

```
data coordinates;
  length
    function
    style
    color    $  8
    position $  1
    size        8
    id          5
    ;
  retain
    xsys
    ysys '2'
    hsys '3'
    when 'a'
    ;
  set coordinates(rename=(name=text));
  function='label';
  style='swissb';
  color='red';
  size=2;
  position='8';
  id=99;
  output;
  function='symbol';
  style='marker';
  text='V';
  color='red';
  size=3;
  id=99;
  output;
run;
```

The Annotate-ready version of the data set `work.coordinates` is shown in Figure 7.



	function	style	color	position	size	id	xsys	ysys	hsys	when	text	description	long	lat
1	label	swissb	red	8	2	99	2	2	3	a	Afsluitdijk Tower	Great for feeding the seagulls	-0.08917	0.924481
2	symbol	marker	red	8	3	99	2	2	3	a	V	Great for feeding the seagulls	-0.08917	0.924481
3	label	swissb	red	8	2	99	2	2	3	a	Bourtange	Can you see Weener from here?	-0.125521	0.925139
4	symbol	marker	red	8	3	99	2	2	3	a	V	Can you see Weener from here?	-0.125521	0.925139
5	label	swissb	red	8	2	99	2	2	3	a	ESTEC	European Space Research and Technology Centre	-0.077148	0.911378
6	symbol	marker	red	8	3	99	2	2	3	a	V	European Space Research and Technology Centre	-0.077148	0.911378
7	label	swissb	red	8	2	99	2	2	3	a	Oosterpoort	Concert venue of the north	-0.114775	0.928757
8	symbol	marker	red	8	3	99	2	2	3	a	V	Concert venue of the north	-0.114775	0.928757
9	label	swissb	red	8	2	99	2	2	3	a	Paradiso	Temple of live indie music	-0.085239	0.913892
10	symbol	marker	red	8	3	99	2	2	3	a	V	Temple of live indie music	-0.085239	0.913892
11	label	swissb	red	8	2	99	2	2	3	a	Tivoli	Place to be in Utrecht	-0.089379	0.909095
12	symbol	marker	red	8	3	99	2	2	3	a	V	Place to be in Utrecht	-0.089379	0.909095
13	label	swissb	red	8	2	99	2	2	3	a	Westerbork Radio Telescope	Anyone out there?	-0.115255	0.923535
14	symbol	marker	red	8	3	99	2	2	3	a	V	Anyone out there?	-0.115255	0.923535
15	label	swissb	red	8	2	99	2	2	3	a	Wieuwerd	We've got mummies!	-0.099299	0.926943
16	symbol	marker	red	8	3	99	2	2	3	a	V	We've got mummies!	-0.099299	0.926943
17	label	swissb	red	8	2	99	2	2	3	a	Yerseke Oyster Banks	Basins full of them...	-0.070753	0.898729
18	symbol	marker	red	8	3	99	2	2	3	a	V	Basins full of them...	-0.070753	0.898729

Figure 7. The Annotate-Ready Version of the `work.coordinates` Data Set

THE RESULTING SAS/GRAPH PLOT

We then append the Annotate data set to the SAS/GRAPH map data set `maps.ntlands`, creating one combined data set, `work.all`, to feed into PROC GPROJECT. Because PROC GPROJECT requires `x` and `y` variables, we rename `long` to `x` and `lat` to `y`:

```
data all;
  set maps.ntlands(drop=x y rename=(long=x lat=y))
      coordinates(rename=(long=x lat=y));
run;
```

Then we use PROC GPROJECT to perform a gnomonic projection of the `x` and `y` variables. We use the `dupok` option to allow duplicate coordinates within an `id` group; otherwise observations would be dropped, notably from the

Annotate part of the data set, where we have two records for each of the placemarks:

```
proc gproject data=all
              out=allp
              project=gnomon
              dupok;
  id id;
run;
```

The output of PROC GPROJECT is the data set `work.allp`, which is identical in structure and content to the input data set `work.all`, except that the values of `x` and `y` have been replaced by their projected values. We then split the projected `work.allp` data set into a new projected map data set, `work.ntlandsp`, and a projected Annotate data set, `work.coordinatesp`. The records belonging to the latter are easily identified by having `id=99`.

```
data ntlandsp coordinatesp;
  set allp;
  if id=99 then output coordinatesp;
  else output ntlandsp;
run;
```

At long last we can plot and annotate. The resulting chart is shown in Figure 8.

```
proc gmap data=ntlandsp map=ntlandsp all;
  id id;
  choro segment / nolegend
                  annotate=coordinatesp;
  run;
quit;
```

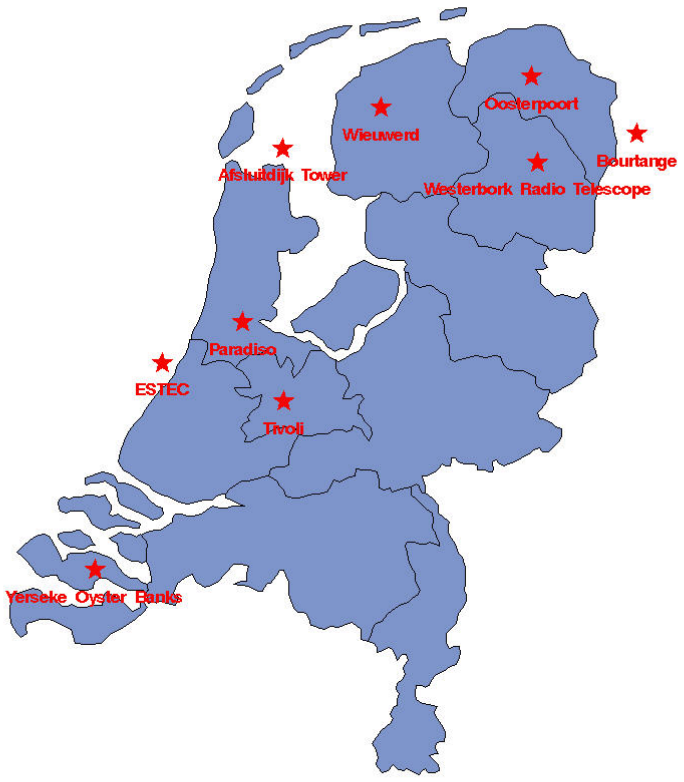


Figure 8. Google Earth Placemarks Plotted with SAS/GRAPH on a Map of the Netherlands.

Interesting result, but not quite what we'd expected. Comparing this with Figure 1, we see that ESTEC should definitely not be somewhere in the North Sea, nor should Bourtange appear in Germany. In fact, all of the other points are also located somewhat off their true positions. How come? A close comparison of the numerical values of the Google Earth coordinates with the unprojected coordinates in the SAS/GRAPH map data set leads us to suspect that the map data set `maps.ntlands`, which dates from 1996, is not exactly the most accurate digital map of the Netherlands. Experiments with more recent map data sets, such as `maps.germany` (data status 2004) and `maps.us` (modified 2007), do not show the offset as observed in Figure 8.

In order to produce a better looking picture than the one in Figure 8, we decided to create a new map data set for the Netherlands by calibrating the existing one by means of Google Earth data. This process is described in the Appendix, since including it here would lead us too far afield. The result of this calibration is a new map data set, `sasdata.ntlands_calibrated`. If you plan to try out code from the paper beyond this point, be sure to create the calibrated map data set first by running the code presented in the Appendix. Submitting the previous code, but with `sasdata.ntlands_calibrated` substituted for `maps.ntlands`, yields a much more palatable result, as shown in the following figure:

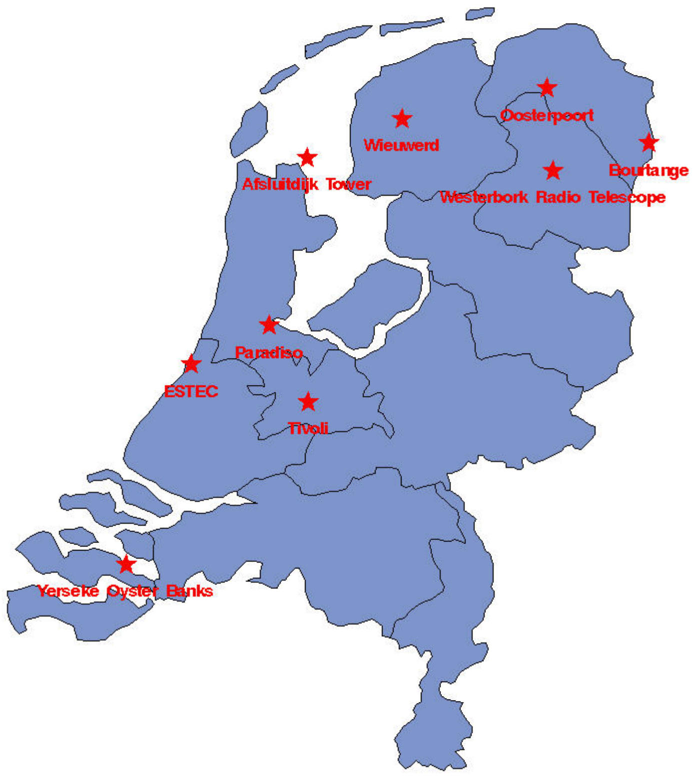


Figure 9. Google Earth Placemarks Plotted with SAS/GRAPH on a Calibrated Map of the Netherlands

RETURN TRIP: FROM SAS TO GOOGLE EARTH

CREATE A SAS/GRAPH PLOT

For the return trip, we assume that we have some customer data in a SAS data set—such as a company name, its marketing slogan, and the location of its head office—as spherical longitude and latitude coordinates expressed in radians:

```
data sasdata.customer_coordinates;
  length
    name        $  50
    description $ 100
    long
    lat            8
    ;
  infile cards dlm=',';
  input
    name
    description
    long
    lat
    ;
  datalines;
Apenheul Inc.,We conduct monkey business...,-0.103306143,0.9113226264
Efteling Theme Park,No mice nor ducks,-0.088135622,0.9014602815
Loo Palace Inn,William of Orange's own Versailles,-0.1037756,0.9116695283
Megaliths R Us,For all your prehistoric masonry needs,-0.118456269,0.9214658393
Neolithic Tombs ACME,Drive-thru and take-away,-0.118637856,0.9238060914
Trenches of Maastricht,Designer trenches by Vauban,-0.099114933,0.8875579212
;
run;
```

For the sake of convenience, we have used the same names for the variables in the `sasdata.customer_coordinates` data set as we did in the first part of the paper: `name`, `description`, `long`, and `lat`. These names enable us to reuse the code to create a SAS/GRAPH plot with only minor modifications:

```
data customer_coordinates;
  length
    function
    style
    color    $  8
    position $  1
    size        8
    id          5
    ;
  retain
    xsys
    ysys '2'
    hsys '3'
    when 'a'
    ;
  set sasdata.customer_coordinates(rename=(name=text));
  function='label';
  style='swissb';
  color='red';
  size=2;
  position='E';
  if text=:'Neolithic' then position=2;
  if text=:'Loo' then position=2;
  id=99;
  output;
  function='symbol';
  style='marker';
  text='V';
  color='red';
  size=3;
  id=99;
  output;
run;

data all;
  set sasdata.ntlands_calibrated(drop=x y rename=(long=x lat=y))
      customer_coordinates(rename=(long=x lat=y));
run;

proc gproject data=all
              out=allp
              project=gnomon
              dupok;
  id id;
run;
```

```
data ntlandsp coordinatesp;
  set allp;
  if id=99 then output coordinatesp;
  else output ntlandsp;
run;

proc gmap data=ntlandsp map=ntlandsp all;
  id id;
  choro segment / nolegend
                  annotate=coordinatesp;
  run;
quit;
```

Note that we use our calibrated map `sasdata.ntlands_calibrated`. We have no reason not to use it, since we know it is more realistic and it already exists. The resulting SAS/GRAPH chart is shown in Figure 10.

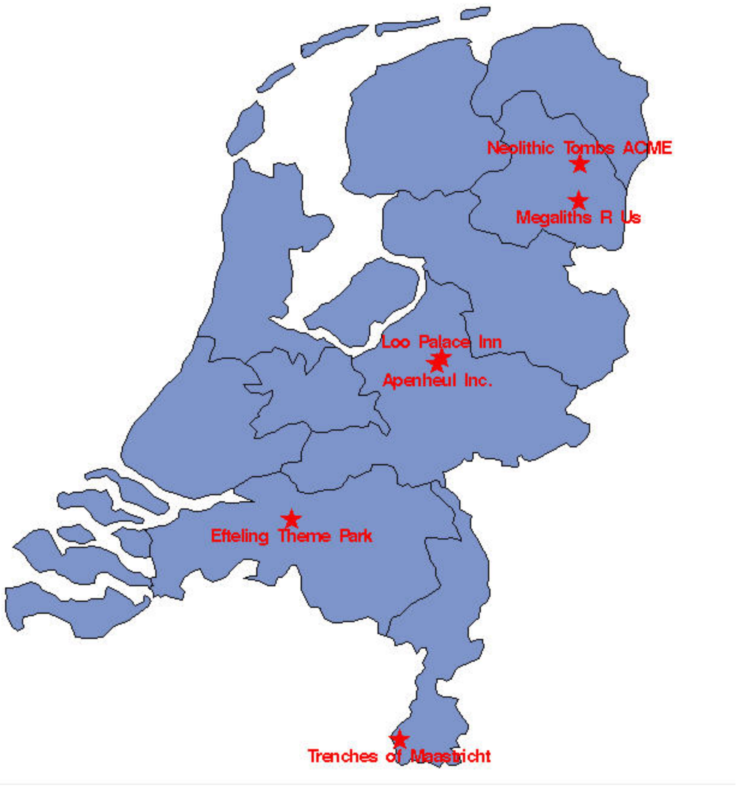


Figure 10. Some SAS Data Plotted with SAS/GRAPH on a Calibrated Map of the Netherlands

TRANSFORM PLOT DATA FOR GOOGLE EARTH USE

We prepare the data set `sasdata.customer_coordinates` for writing to an XML file that is compatible with Google Earth by converting radians into degrees and putting the converted longitude and latitude into a comma-separated string. It is not necessary to add a zero altitude for Google Earth to interpret the coordinates string correctly. The resultant data set is `work.placemark`:

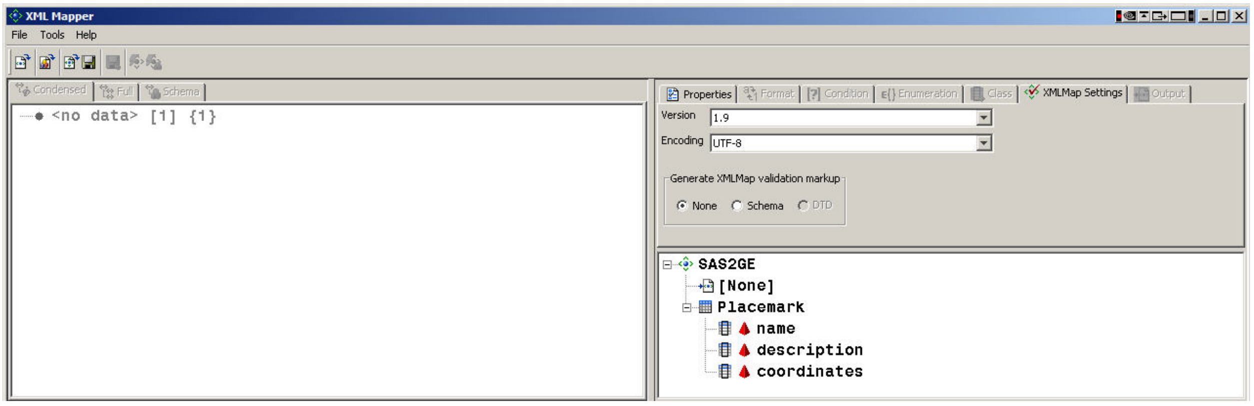
```
data placemark(keep=name description coordinates);
  set sasdata.customer_coordinates;
  length
    pi
    long_deg
    lat_deg          8
    coordinates $ 100
    ;
  pi=constant('pi');
  long_deg=-long*180/pi;
  lat_deg=lat*180/pi;
  coordinates=strip(put(long_deg,best.))||','||strip(put(lat_deg,best.));
run;
```

USE SAS XML MAPPER TO CREATE AN EXPORT XML MAP

SAS 9.2 comes with a new XML LIBNAME Engine, xml92. The xml92 engine supports version 1.9 of the SXLE Map Syntax, which provides new tags for writing XML files via an XML Map. In previous versions of SAS, the xml engine was suitable only for reading XML files by means of an XML Map, and if one wanted to write XML, the only way to generate a specific XML layout was by coding a lot of PUT statements. Things have gotten easier!

We first need to create a version 1.9 XML Map suitable for export purposes. Luckily, we can reuse our work from the first part of the paper where we created the map `ge2sas.map`. We start the SAS XML Mapper application, select **`Open XML Map`** from the **`File`** menu, and load our map file `ge2sas.map`.

We change the map name from `GE2SAS` to `SAS2GE`, and on the **`XML Map Settings`** tab we change the version from 1.2 to 1.9 (Figure 11). Note that an output item named `[None]` is automatically added to the map.


XML Mapper
File Tools Help
Condensed Full Schema
Properties Format Condition Enumeration Class XMLMap Settings Output

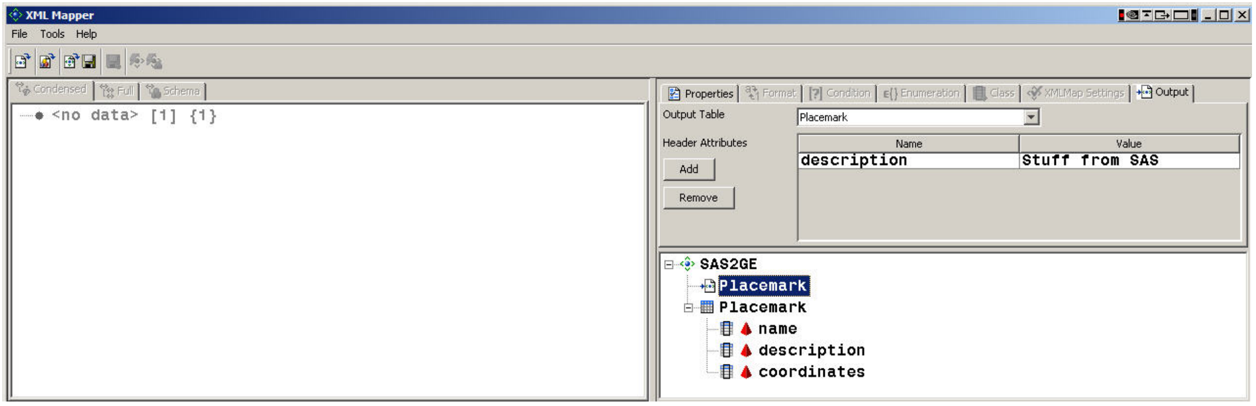
```
<no data> [1] {1}
```

Version: 1.9
Encoding: UTF-8
Generate XMLMap validation markup: [✓ None] Schema DTD

```
SAS2GE
  [None]
  Placemark
    name
    description
    coordinates
```

Figure 11. Changing the XML Map Syntax Version to 1.9

When we select the output item `[None]`, the **`Output`** tab becomes available. On the **`Output`** tab, we enter `Placemark` as the name of the output table. At least one header attribute must be added, so we enter `description` for the attribute name, and `Stuff from SAS` for the attribute value. (Figure 12)


XML Mapper
File Tools Help
Condensed Full Schema
Properties Format Condition Enumeration Class XMLMap Settings Output
Add
Remove

```
<no data> [1] {1}
```

Output Table: Placemark

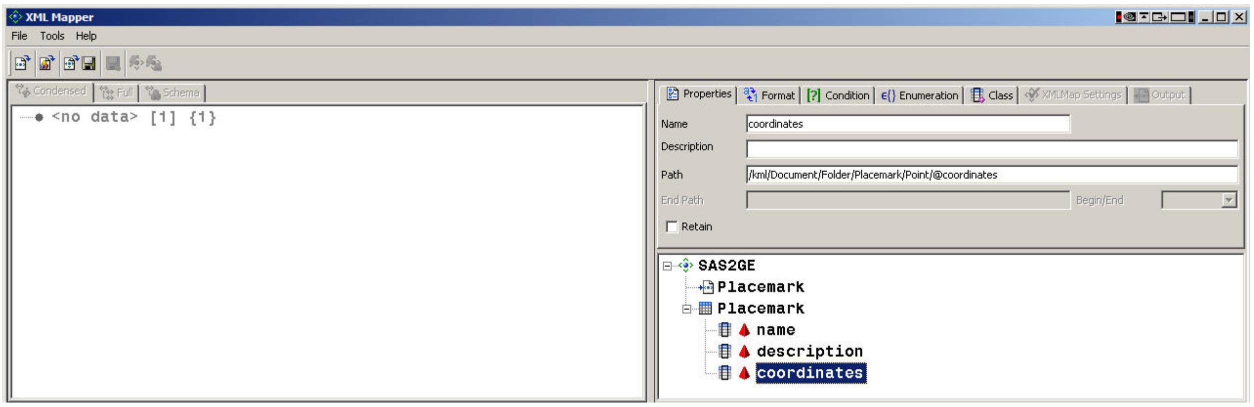
Header Attributes

Name	Value
description	Stuff from SAS

```
SAS2GE
  Placemark
  Placemark
    name
    description
    coordinates
```

Figure 12. Entering the Table Name and Header Attribute

We then select the `coordinates` column in the Map pane and go to the **`Properties`** tab, where we change the path from `/kml/Document/Folder/Placemark/Point/coordinates` to `/kml/Document/Folder/Placemark/Point/@coordinates` (Figure 13). The added @ in front of the column name causes the coordinates data to be written as a name-value pair within the `Point` tag, and provides a way of preserving the required nesting of tags in the Google Earth KML layout.



```
<no data> [1] {1}
```

Name: coordinates
Description:
Path: /kml/Document/Folder/Placemark/Point/@coordinates
End Path:

```
SAS2GE
  Placemark
  Placemark
    name
    description
    coordinates
```

Figure 13. Adding an @ to the Xpath of the Coordinates Column

We then select **`Save XML Map As`** from the **`File`** menu, and save our export map as `sas2ge.map` in our XML Maps directory. Here is the new XML Map:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ############################################################ -->
<!-- 2007-12-31T14:15:51 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 9.2.0.000000_v920c_20071205_7927 -->
<!-- ############################################################ -->
<!-- ###  Validation report                                   ### -->
<!-- ############################################################ -->
<!-- XMLMap validation completed successfully. -->
<!-- ############################################################ -->
<SXLEMAP name="SAS2GE" version="1.9">
  <!-- ############################################################ -->
  <OUTPUT>
    <HEADING>
      <ATTRIBUTE name="description" value="Stuff from SAS"/>
    </HEADING>
    <TABLEREF name="Placemark"/>
  </OUTPUT>
  <!-- ############################################################ -->
  <TABLE name="Placemark">
    <TABLE-PATH syntax="XPath">/kml/Document/Folder/Placemark</TABLE-PATH>
    <COLUMN name="name">
      <PATH syntax="XPath">/kml/Document/Folder/Placemark/name</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>50</LENGTH>
    </COLUMN>
    <COLUMN name="description">
      <PATH syntax="XPath">/kml/Document/Folder/Placemark/description</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>100</LENGTH>
    </COLUMN>
    <COLUMN name="coordinates">
      <PATH syntax="XPath">/kml/Document/Folder/Placemark/Point/@coordinates</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>100</LENGTH>
    </COLUMN>
  </TABLE>
</SXLEMAP>
```

WRITE A GOOGLE EARTH XML FILE USING THE XML MAP

Writing a Google Earth KML file is now simple. We define filerefs to the output KML file (no point in making an XML file and then having to rename it with a .kml extension) and to the export XML Map:

```
filename out 'c:\ge2sas2ge\xml output\placemarks from sas.kml';
filename s2gmap 'c:\ge2sas2ge\xml maps\sas2ge.map';
```

The output LIBNAME `out` is defined using the new xml92 engine and the export XML Map `s2gmap`:

```
libname out xml92 xmlmap=s2gmap;
```

Creating XML output is then as simple as this:

```
data out.Placemark;
  set placemark;
run;
```

At this point it is very important to remember that, like everything in XML and XML Maps, table names are case sensitive! In the previous section, we named the output table `Placemark`, with a capital P. Therefore, the capital P in the above DATA step is absolutely essential. If we were to submit the same DATA step, but with `out.placemark` instead of `out.Placemark`, we would get the message “ERROR: XMLMap for output table placemark requested but that table did not exist in the XMLMap output section.”

Here is the generated XML file `placemarks from sas.kml`:

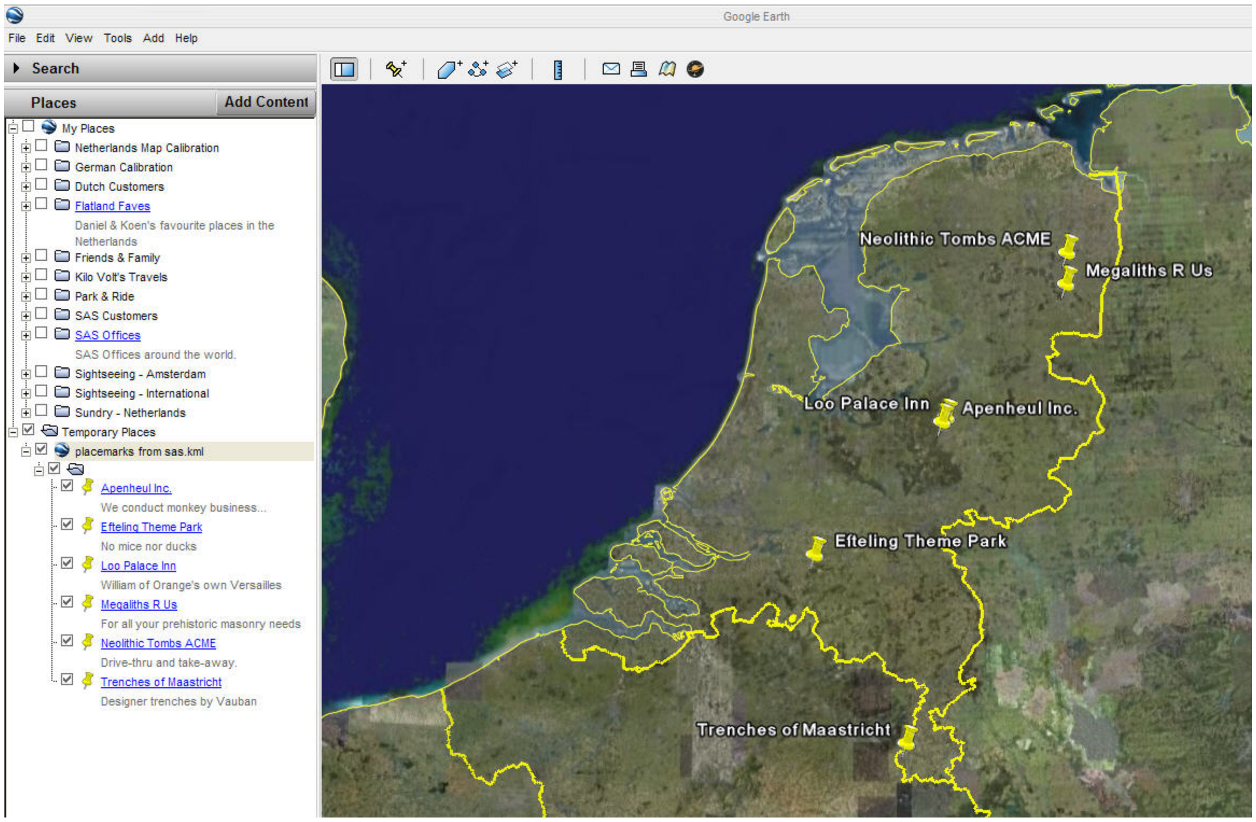
```
<?xml version="1.0" encoding="windows-1252" ?>
<!--
      SAS XML Libname Engine (SAS92XML)
      SAS XMLMap Generated Output
      Version 9.02.01B0P12042007
      Created 2008-01-04T22:36:33
  -->
<kml description="Stuff from SAS">
   <Document>
      <Folder>
         <Placemark>
            <name>Apenheul Inc.</name>
            <description>We conduct monkey business...</description>
            <Point coordinates="5.9190059917,52.214940267" />
         </Placemark>
         <Placemark>
            <name>Efteling Theme Park</name>
            <description>No mice nor ducks</description>
            <Point coordinates="5.0497991654,51.649869529" />
         </Placemark>
         <Placemark>
            <name>Loo Palace Inn</name>
            <description><![CDATA[William of Orange's own
              Versailles]]></description>
            <Point coordinates="5.9459038964,52.234816282" />
         </Placemark>
         <Placemark>
            <name>Megaliths R Us</name>
            <description>For all your prehistoric masonry needs</description>
            <Point coordinates="6.7870442706,52.796103557" />
         </Placemark>
         <Placemark>
            <name>Neolithic Tombs ACME</name>
            <description>Drive-thru and take-away.</description>
            <Point coordinates="6.7974484393,52.930190126" />
         </Placemark>
         <Placemark>
            <name>Trenches of Maastricht</name>
            <description>Designer trenches by Vauban</description>
```

```
                <Point coordinates="5.6788673476,50.853322958" />
            </Placemark>
        </Folder>
    </Document>
</kml>
```

Note the way the coordinates are written inside the `Point` tag thanks to the @ added to the coordinates' Xpath in the previous section.

IMPORT XML AS GOOGLE EARTH PLACEMARKS

And now the final test: we can either double-click the `placemarks from sas.kml` file that we generated from SAS, or open it from within Google Earth. Figure 14 shows the result in Google Earth, and this is exactly what we were trying to achieve.



My Places
Netherlands Map Calibration
German Calibration
Dutch Customers
Flatland Faves
Daniel & Koen's favourite places in the Netherlands
Friends & Family
Kilo Volt's Travels
Park & Ride
SAS Customers
SAS Offices
SAS Offices around the world.
Sightseeing - Amsterdam
Sightseeing - International
Sundry - Netherlands
Temporary Places
placemarks from sas.kml
Apenheul Inc.
We conduct monkey business...
Efteling Theme Park
No mice nor ducks
Loo Palace Inn
William of Orange's own Versailles
Megaliths R Us
For all your prehistoric masonry needs
Neolithic Tombs ACME
Drive-thru and take-away.
Trenches of Maastricht
Designer trenches by Vauban

Figure 14. Mission Accomplished: Data from SAS is Transformed into Google Earth Placemarks

SUMMARY

We have shown how to export data about geographical locations—placemarks—from Google Earth and import them into SAS by means of an XML Map, which can be reused for any set of Google Earth placemarks. We massaged the data and plotted it on a SAS/GRAPH map. In the process of doing this, we created a more realistic map data set for the Netherlands than the one that is currently available with SAS 9.2. Going the other way, we created a new XML Map for exporting data from SAS and generated a set of Google Earth placemarks based on some sample geographic data which we used to make a SAS/GRAPH plot.

The two XML Maps made in this paper only reference a small number of all the possible data elements available in the Google Earth KML specification. An export XML Map with more bells and whistles can be constructed. For example, one can include KML `Style` definitions to completely customize the look and feel of the exported placemarks, manage the content of the placemarks' balloons, specify individual viewing options via the `LookAt` element, and much more. We leave this as an exercise to the reader and look forward to future contributions using this technique to good effect in the realm of data visualization.

REFERENCES

Davis, Michael. 1997. “Putting Yourself on the Map with the GMAP Procedure.” *Proceedings of the Twenty-Second Annual SAS Users Group International Conference*, San Diego, CA, 274–283. Cary, NC: SAS Institute, Inc. Available at: http://www.bassettconsulting.com/PAPER49.PDF.

Google. 2007. “KML Documentation Introduction.” http://code.google.com/apis/kml/documentation/.

Mintz, David. 2008. “Using SAS and Google Earth to Access and Display Air Pollution Data.” *Proceedings of the SAS Global Forum 2008 Conference*, San Antonio, TX.

SAS Institute Inc. 2007. SAS Customer Support site. “XML Map Syntax Version 1.2.” http://support.sas.com/onlinedoc/913/getDoc/en/engxml.hlp/a002484805.htm.

ACKNOWLEDGMENTS

The authors would like to thank all their SAS colleagues who made helpful suggestions during the writing of the paper, and especially the participants of the 2006 SAS Forum Netherlands for inspiring us to find a way to pull this off.

CONTACT INFORMATION

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

Daniël Kuiper / Koen Vyverman
SAS Institute, the Netherlands
Flevolaan 69
1272PC Huizen
The Netherlands
Email: support@snl.sas.com

APPENDIX: USING GOOGLE EARTH DATA TO CALIBRATE THE MAPS.NTLANDS SAS MAP DATA SET

In an effort to obtain a better SAS/GRAPH map data set for the Netherlands, we use the approach of calibrating the existing map data set `maps.ntlands` by means of data extracted from Google Earth. In Google Earth, we scan the border of the Netherlands for those points having minimum longitude, maximum longitude, minimum latitude, and maximum latitude. This yields four placemarks, West, East, North, and South. (Figure 15).

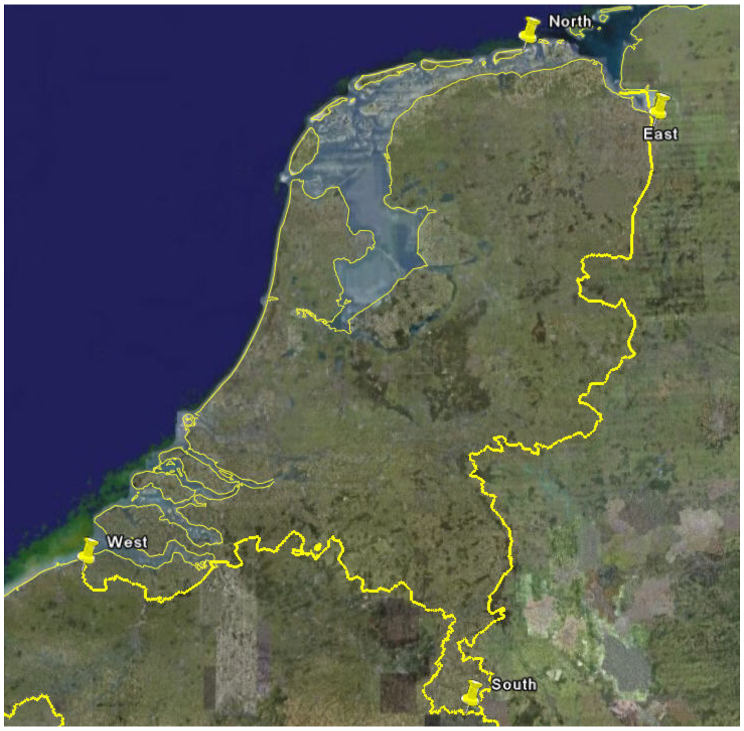


Figure 15. Google Earth placemarks for the most western, eastern, northern, and southern points in the Netherlands.

We put these in a folder which we save as the file `netherlands map calibration.kml` in our XML Input directory, and we change the extension to .xml.

The idea is to look in the `maps.ntlands` data set and likewise identify the four extremes of latitude and longitude. Once these have been found, a simple mapping can be constructed to transform the points in `maps.ntlands` in such a fashion that its longitude and latitude extremes coincide with those obtained from Google Earth. This transformation is basically a translation (to make the mid-point of the two maps coincide) followed by a scaling to stretch (or shrink) the new map in two directions.

The code below does all this, and produces a calibrated map data set `sasdata.ntlands_calibrated`, which we use a number of times in the paper.

```
/* Read the calibration points from Google Earth into SAS.          */

filename gedata 'c:\ge2sas2ge\xml input\netherlands map calibration.xml';
filename g2smap 'c:\ge2sas2ge\xml maps\ge2sas.map';
libname gedata xml xmlmap=g2smap access=readonly;
libname sasdata 'c:\ge2sas2ge\sas data';

data calibration;
  set gedata.placemark;
run;

libname gedata clear;
filename gedata clear;
filename g2smap clear;
```

```
  data calibration(drop=delimiters modifiers coordinates pi);
  set calibration;
  length
    delimiters $ 1
    modifiers  $ 2
    pi
    long
    lat  8
    ;
  pi=constant('pi');
  delimiters=',';
  modifiers='r';
  long=-pi/180*input(scan(coordinates,1,delimiters,modifiers),20.);
  lat=pi/180*input(scan(coordinates,2,delimiters,modifiers),20.);
run;

/* Stuff the minima and maxima of the Google Earth coordinates into */
/* some macro variables.                                            */

data _null_;
  set calibration;
  if upcase(name)=upcase('east') then
    call symput('ge_min_long',strip(put(long,best32.)));
  if upcase(name)=upcase('west') then
    call symput('ge_max_long',strip(put(long,best32.)));
  if upcase(name)=upcase('north') then
    call symput('ge_max_lat',strip(put(lat,best32.)));
  if upcase(name)=upcase('south') then
    call symput('ge_min_lat',strip(put(lat,best32.)));
run;

/* Same for the minima and maxima of LONG and LAT coordinates in    */
/* the maps.ntlands data set.                                       */

proc sql noprint;
  select
    strip(put(min(long),best32.)),
    strip(put(max(long),best32.)),
    strip(put(min(lat),best32.)),
    strip(put(max(lat),best32.))
  into
    :sas_min_long,
    :sas_max_long,
    :sas_min_lat,
    :sas_max_lat
  from
    maps.ntlands
  ;
quit;

/* We translate the midpoint of the SAS map to the origin of the    */
/* coordinate system, then we scale longitude and latitude accor-   */
/* ding to the size of the Google Earth map, and finally translate  */
/* the SAS map back so its midpoint coincides with the midpoint of  */
/* the Google Earth map.                                            */

data sasdata.ntlands_calibrated(drop=
                                     ge_mid_long
                                     ge_mid_lat
                                     sas_mid_long
                                     sas_mid_lat
                                     long_stretch_factor
                                     lat_stretch_factor
                               );
  set maps.ntlands;
  length
    ge_mid_long
    ge_mid_lat
    sas_mid_long
    sas_mid_lat
    long_stretch_factor
    lat_stretch_factor   8
    ;
  retain
    ge_mid_long
    ge_mid_lat
    sas_mid_long
    sas_mid_lat
    long_stretch_factor
    lat_stretch_factor
    ;
  if _n_=1 then do;
    ge_mid_long=(&ge_max_long+&ge_min_long)/2;
    ge_mid_lat=(&ge_max_lat+&ge_min_lat)/2;
    sas_mid_long=(&sas_max_long+&sas_min_long)/2;
    sas_mid_lat=(&sas_max_lat+&sas_min_lat)/2;
    long_stretch_factor=abs(&ge_max_long-&ge_min_long)/
                        abs(&sas_max_long-&sas_min_long);
    lat_stretch_factor=abs(&ge_max_lat-&ge_min_lat)/
                       abs(&sas_max_lat-&sas_min_lat);
    end;
  long=ge_mid_long+long_stretch_factor*(long-sas_mid_long);
  lat=ge_mid_lat+lat_stretch_factor*(lat-sas_mid_lat);
run;
```

The effect of using the calibrated map can be seen by plotting the four placemarks West, East, North, and South on the old and new maps (Figure 16).

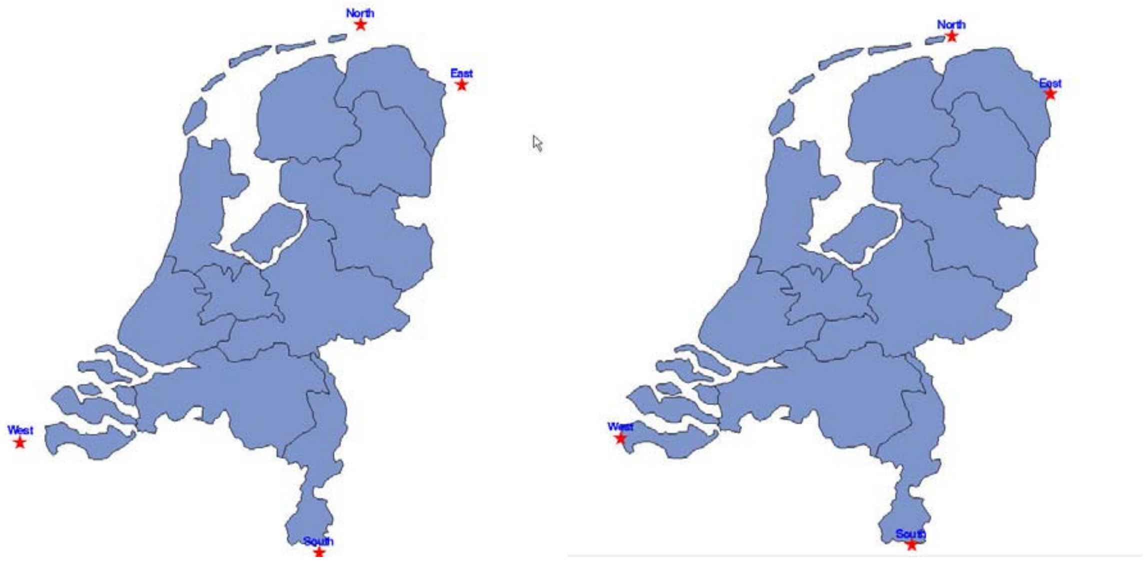


Figure 16. Google Earth Placemarks for the Most Western, Eastern, Northern, and Southern points in the Netherlands Plotted on the Standard SAS/GRAPH Map (left) and the Calibrated Map (right)