Paper 131-2010

Mashups Can Be Gravy: Techniques for Bringing the Web to SAS®

Jack Fuller, COMSYS LLC, Portage, MI

ABSTRACT

A mashup is an agglomeration of disparate content. While popular terminology tends to associate mashups with web-based data, physician John Snow demonstrated an early example of mashing content in 1854 when he identified the source of a London cholera outbreak by mapping cholera patients with their water sources. The guilty culprit turned out to be an infected water pump handle in Soho. Web based mashups have transitioned from metasearch engines like Dogpile® and Webcrawler® to Google™ maps detailing winery tours and public restrooms to Yahoo!® Pipes™ which allows users to define their own mashups.

SAS® has long facilitated moving information from SAS to the Web. This paper will explore mashup techniques for enabling the movement of information in the other direction: from the Web to SAS.

INTRODUCTION

I first became interested in accessing web-based information while browsing the web and reading different financial analyses. Most of these would perform some analysis and then discuss the results. I wondered why I couldn't do my own analyses using SAS analytics. The specific analysis which precipitated my inquiry involved comparing stock prices to their historical growth rates. Finding a method for actually bringing the data into SAS presented my first and most obvious obstacle.

This paper presents real world examples, from the simple to the complex, to explore various techniques which I have used to bring Web based data into SAS: first, by accessing structured text data such as Comma Separated Values (CSV) and Extensible Markup Language (XML) files; second, by accessing unstructured text data such as HTML documents; and finally, by bringing everything together with an example which includes retrieving local tax records, computing some descriptive statistics and then displaying the results using Google Earth.

ACCESSING STRUCTURED TEXT DATA

One of the most straightforward methods for accessing Web based information is to simply access structured text files. These can generally be brought into SAS with minimal coding.

RETRIEVING HISTORICAL CSV STOCK QUOTES FROM YAHOO! FINANCE

The key points for accessing web based CSV data can be summarized as:

- Specify a FILENAME statement with the URL option which points to the appropriate web URL
- Specify a comma delimiter for the FILENAME statement in a data step

For example, the following code retrieves CSV monthly historical stock quotes General Electric (GE) from Yahoo (note that the adjusted closing price (adjClose) has already taken into account stock splits and dividends):

```
filename foo url 'http://ichart.finance.yahoo.com/table.csv?s=GE&g=m';

data WORK.quotes;
  length ticker $5 date open high low close volume adjClose 8;
  infile foo delimiter=',' firstobs=2;
  input date : yymmdd10. open high low close volume adjClose;
  format date mmddyy10.;
  retain ticker "GE";
run;

filename foo;
```

Running the above code creates the data set shown in Figure 1.

VIEWTABLE: Work.Quotes

	ticker	date	open	high	low	close	volume	adjClose
1	GE	10/01/2009	16.31	16.87	15.11	15.34	104290500	15.34
2	GE	09/01/2009	13.74	17.52	13.03	16.42	132466400	16.42
3	GE	08/03/2009	13.65	14.88	13.16	13.9	86516200	13.82
4	GE	07/01/2009	11.76	13.45	10.5	13.4	100768400	13.32
5	GE	06/01/2009	13.82	13.99	11.25	11.72	87259400	11.65
6	GE	05/01/2009	12.74	14.55	12.22	13.48	104569000	13.29
7	GE	04/01/2009	9.91	12.81	9.8	12.65	133547800	12.47
8	GE	03/02/2009	8.29	11.35	5.87	10.11	277426300	9.97
9	GE	02/02/2009	12.03	12.9	8.4	8.51	194928800	8.39

Figure 1
GE Historical Stock Quotes from Yahoo!

We can then use SAS code to address the original question of how the actual price compares to its predicted price (see Figure 2). Further investigations could address questions such as: what is the expected return if the price does revert to its predicted value?

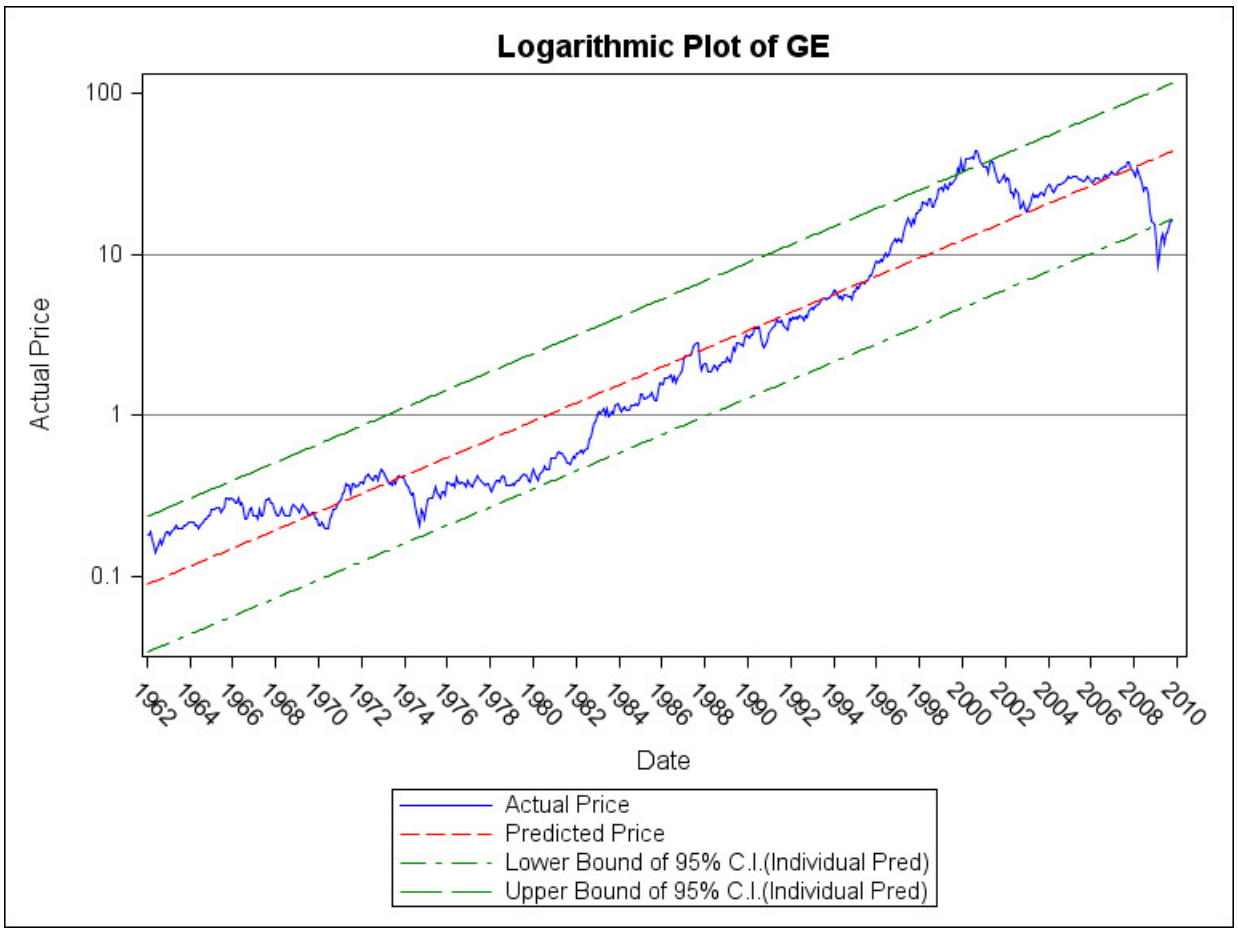


Figure 2
Time Series Logarithmic Plot of GE Stock's Price (generated using PROC SGPLOT)

XML FILES: USING YAHOO! WEB SERVICES FOR GEOCODING

The key points for accessing web based XML data can be summarized as:

1. Create an XML map which describes the XML format
2. Specify a FILENAME which points to the XML map
3. Specify a FILENAME which uses the URL access method and points to the URL which contains the XML data
4. Specify a LIBNAME with the same name as the FILENAME which points to the XML data and which uses the XML access method and XMLMAP option

For example, the following XML map describes the XML which is returned by Yahoo! web services when geocoding an address for latitude and longitude.

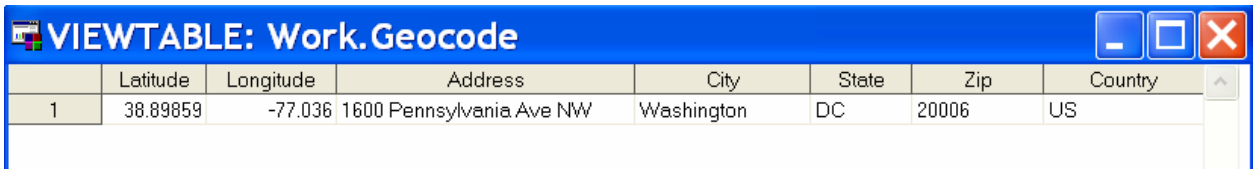
```
<?xml version="1.0" encoding="windows-1252"?>

<!-- ############################################################ -->
<!-- 2009-08-20T13:21:09 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 902000.3.6.20090116170000_v920 -->
<!-- ############################################################ -->
<!-- ###  Validation report                                   ### -->
<!-- ############################################################ -->
<!-- XMLMap validation completed successfully. -->
<!-- ############################################################ -->
<SXLEMAP name="yahooGeoMap" version="1.2">

  <!-- ############################################################ -->
  <TABLE name="Result">
    <TABLE-PATH syntax="XPath">/ResultSet/Result</TABLE-PATH>

    <COLUMN name="Latitude">
      <PATH syntax="XPath">/ResultSet/Result/Latitude</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>double</DATATYPE>
    </COLUMN>

    <COLUMN name="Longitude">
      <PATH syntax="XPath">/ResultSet/Result/Longitude</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>double</DATATYPE>
    </COLUMN>

    <COLUMN name="Address">
      <PATH syntax="XPath">/ResultSet/Result/Address</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>32</LENGTH>
    </COLUMN>

    <COLUMN name="City">
      <PATH syntax="XPath">/ResultSet/Result/City</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>32</LENGTH>
    </COLUMN>

    <COLUMN name="State">
      <PATH syntax="XPath">/ResultSet/Result/State</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>2</LENGTH>
    </COLUMN>

    <COLUMN name="Zip">
      <PATH syntax="XPath">/ResultSet/Result/Zip</PATH>
```

```
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>10</LENGTH>
    </COLUMN>

    <COLUMN name="Country">
      <PATH syntax="XPath">/ResultSet/Result/Country</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>2</LENGTH>
    </COLUMN>

  </TABLE>

</SXLEMAP>
```

In order to use Yahoo! web services, it is necessary to have an APPID token. This can be obtained by creating a Yahoo! account and then requesting one. Google uses a similar methodology for allowing access to their web services. In order to run the following code, you will need to replace highlighted <appID> with a valid token:

```
filename sxlemap 'C:\Projects\yahooGeoXMLMap.map';

filename foo url
'http://local.yahooapis.com/MapsService/V1/geocode?appid=<appID>&output=xml&city=WAS
HINGTON&state=DC&street=1600%20PENNSYLVANIA%20AVE';

libname foo xml xmlmap=sxlemap access=READONLY;

data WORK.geocode;
  set FOO.result;
run;

libname foo;

filename foo;
```

Running the previous code produces a data set with geocoding information for 1600 Pennsylvania Avenue in Washington, DC (see Figure 3).



	Latitude	Longitude	Address	City	State	Zip	Country
1	38.89859	-77.036	1600 Pennsylvania Ave NW	Washington	DC	20006	US

Figure 3
Geocoding Information Retrieved From Yahoo! Web Services

Both the CSV and XML files consist of text with a known structure. The next level of complexity consists of text files where the structure needs to be specified.

ACCESSING UNSTRUCTURED TEXT DATA

The key points for accessing unstructured text data can be summarized as:

- Examine the underlying text (e.g. HTML) to determine an appropriate pattern
- Use Perl to actually parse the HTML with a Regular Expression
- Specify a FILENAME statement with the PIPE option which points to a Perl program which will do the work of parsing the HTML
- Specify a delimiter when using the FILEREF in a data step

Working with Web based unstructured text files can be much more difficult because it essentially involves screen scraping (i.e. parsing) the desired text from HTML source code. The first step is to examine the HTML source code and look for a pattern. The following example is from Yahoo! Finance's key statistics page for GE (blue highlighting indicates text that we want to match, violet highlighting indicates text that we want to capture):

```
<tr>
  <td class="yfnc_tablehead1" width="75%">
    Market Cap (intraday)<font size="-1"><sup>5</sup></font>:
  </td>
  <td class="yfnc_tabledata1">
    <span id="yfs_j10_ge">
      170.88B
    </span>
  </td>
</tr>
```

Essentially, we want to parse table cells which have a cascading style sheet class of "yfnc_tablehead1" for the descriptions and table cells which have a cascading style sheet class of "yfnc_tabledata1" for the actual data. We could use the following regular expression to parse the information from the text (violet highlighting indicates the portion of the regular expression text used for buffer capturing):

```
<td.+?class=\"yfnc_tablehead1\"[^>]*?>([^<:\n]*).+?<td.+?class=\"yfnc_tabledata1\"[^
>]*?>(?:<span.*?>)?([^<:\n]*)
```

- "(?:" is used to specify grouping without capturing: e.g. "(?:<span.*?)"
- "?" must be used to ensure lazy evaluation of some of the quantifiers: e.g. "<td.+?"

However, the problem with simply looping through the HTML file and applying the regular expression is that SAS wants to retrieve the HTML line by line and there is no way of knowing which portion of the regular expression is on which line. It could consist of ten lines (as in the example) or it could all be on one line. What is needed is a method of applying our regular expression across newline boundaries.

LEVERAGING PERL

One solution is to leverage Perl to process the HTML text prior to its entry to the SAS system. The key idea is that Perl can provide access to a greater set of regular expression modifiers which can then be used to handle newline boundaries:

- m modifier : treat the start and end of line anchors so that they can match anywhere in the string
- s modifier : treat "." as matching any character (including a newline)

The following Perl program takes two arguments:

- the URL to use as input
- the regular expression which will be applied to the URL

It then cycles through the URL while returning the capture buffers of all of the matches as tab delimited text:

```
#!/usr/bin/perl -w
use LWP::Simple;

# Ensure the correct number of arguments
die "Incorrect number of command line arguments\n" if (@ARGV ne 2);

# Retrieve the url
my $url = get($ARGV[0]) || die "Unable to open url\n";

# Loop thru the url
while ($url =~ m{$ARGV[1]}gsmi) {

  my $sep = "";

  # Loop thru the capture buffers
  for (my $i=1; $i <= $#-; $i++) {
```

```
      print "$sep" . "${$i}";
      $sep = "\t";

    }

    print "\n";

  }
```

RETRIEVING KEY FINANCIAL STATISTICS FROM YAHOO!

If we save the previous Perl program in "C:\Projects\screenScrape.pl", we can then use the following code to retrieve the key statistics for GE from Yahoo! finance:

```
filename foo pipe
'C:\Projects\screenScrape.pl "http://finance.yahoo.com/q/ks?s=GE"
"<td.+?class=\"yfnc_tablehead1\"[^>]*?>([^<:\n]*).+?<td.+?class=\"yfnc_tabledata1\"[
^>]*?>(?:<span.*?>)?([^<:\n]*)"';

data WORK.keyStats;
  length stat value $64;
  infile foo dsd dlm='09'x truncover;
  input stat value;
run;

filename foo;
```

This returns the data set in Figure 4.

VIEWTABLE: Work.Keystats

	stat	value
1	Market Cap (intraday)	161.70B
2	Enterprise Value (23-Oct-09)	567.29B
3	Trailing P/E (ttm, intraday)	14.04
4	Forward P/E (fye 31-Dec-10)	16.52
5	PEG Ratio (5 yr expected)	1.48
6	Price/Sales (ttm)	1.01
7	Price/Book (mrq)	1.39
8	Enterprise Value/Revenue (ttm)	3.51
9	Enterprise Value/EBITDA (ttm)	23.34

Figure 4
GE Key Statistics Financial Data from Yahoo!

These techniques for importing data from the Web into SAS can be used in more elaborate mashups where the main limiting factor is not one of inaccessibility but only one of imagination.

FINAL EXAMPLE COMBINING TAX RECORDS, SAS STATISTICS AND GOOGLE EARTH

For our final example, we will create a mashup by pulling a list of tax parcel IDs, querying the tax information for each parcel ID, performing a few simple analytics and then creating a Keyhole Markup Language (KML) document to display the results in Google Earth.

PULLING A LIST OF TAX PARCEL IDS

The first step is to retrieve the list of parcel IDs from the online tax records. The following code leverages our Perl program to return a list of parcel IDs for houses on "Bay Ridge".

```
options noQuoteLenMax;

/* NOTE: "\x22" is used in place of an actual double quote in order to allow more
straightforward quoting */

filename foo pipe
'C:\Projects\screenScrape.pl
"http://www.kalcounty.com/equalization/parcel_search.php?op=search&by=property_addre
ss&search_str=bay%20ridge"
 "<tr\s.+?<td\s+class=\x22text-
small\x22>\s*<a\s+href=\x22[^\x22]+\x22>\s*(.+?)\s*</a>"';

data WORK.taxInfo;
  length parcelNo $32;
  infile foo dsd dlm='09'x noprint truncover;
  input parcelNo;
run;

filename foo;
```

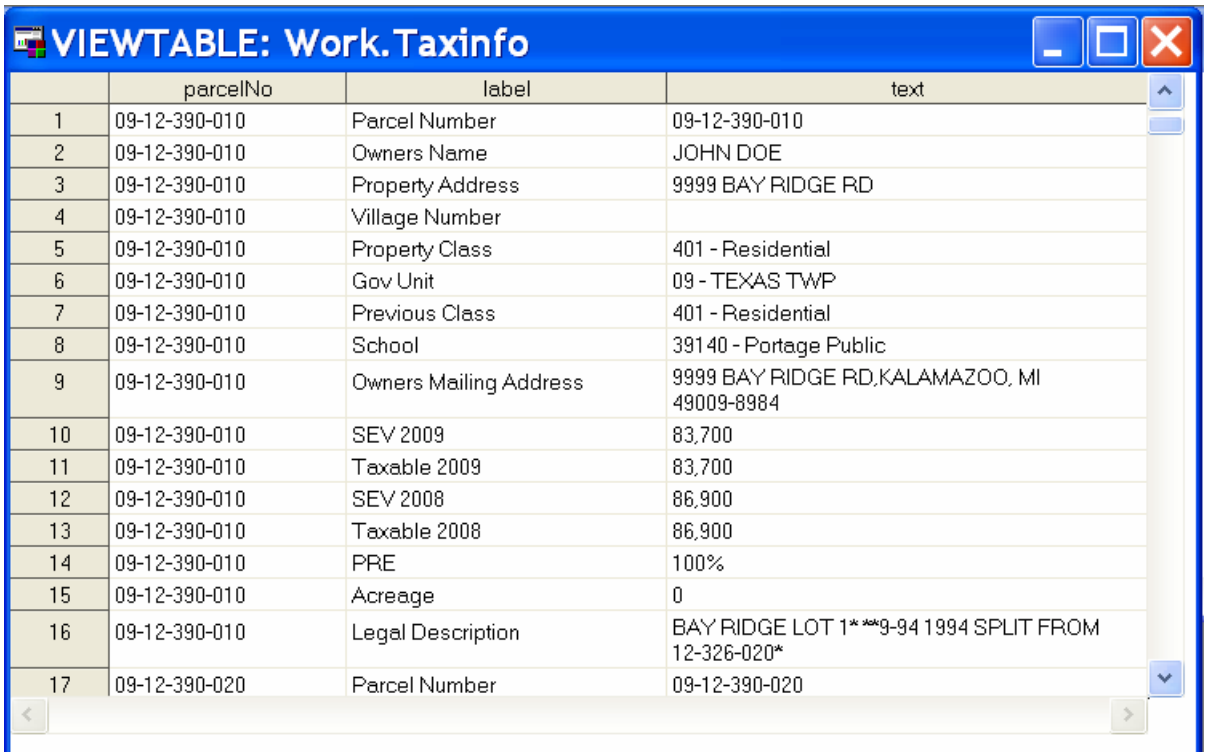
QUERYING THE TAX INFORMATION FOR EACH PARCEL ID

Once we have the list of parcel IDs, we can then query the online records for the specific tax information by using the FILEVAR option on the FILENAME statement.

```
data WORK.taxInfo(drop=_:);
  length _fileVar $256 _year $4;
  retain _year;

  if (_n_=1) then do;
    _prxLblBadChar = prxparse("s/['.]//i");
    _prxLblNBSP = prxparse('s/ / /i');
    _prxLblNum = prxparse('s/^(\d+)\s*(.+?)\s*$/$2 $1/i');
    _prxTxtBR = prxparse('s/<br>/,/i');
    _prxTxtNBSP = prxparse('s/ / /i');
    retain _prx:;
  end;

  set WORK.taxInfo;
  length label $32 text $64;

  _fileVar = catx(' ', "C:\Projects\screenScrape.pl",
quote(cats("http://www.kalcounty.com/equalization/parcel_detail.php?parcel_id=",
parcelNo)),
quote("<td\s+class=\x22label\x22>\s*(.+?)\s*:?\s*</td>\s*<td\s+class=\x22text\x22.*?
>\s*(.*?)\s*</td>"));
  infile foo pipe filevar=_fileVar dsd dlm='09'x noprint truncover end=_atEnd;

  *** Loop until there are no more label/text pairs ***;
  do while (NOT _atEnd);
    input label text;

    *** Clean the label ***;
    if prxmatch(_prxLblNum, label) then do;
      _year = prxposn(_prxLblNum, 1, label);
      label = prxchange(_prxLblNum, -1, label);
    end;
    else if NOT missing(_year) then do;
      label = catx(' ', label, _year);
      _year = ' ';
    end;
    label = prxchange(_prxLblBadChar, -1, label);
    label = prxchange(_prxLblNBSP, -1, label);
    label = strip(label);

    *** Clean the text ***;
```

```
    text = prxchange(_prxTxtBR, -1, text);
    text = prxchange(_prxTxtNBSP, -1, text);
    text = strip(text);
    OUTPUT;
  end;
run;
```

This produces a data set (see Figure 5) which contains the tax records in a vertical format.



VIEWTABLE: Work.Taxinfo

	parcelNo	label	text
1	09-12-390-010	Parcel Number	09-12-390-010
2	09-12-390-010	Owners Name	JOHN DOE
3	09-12-390-010	Property Address	9999 BAY RIDGE RD
4	09-12-390-010	Village Number	
5	09-12-390-010	Property Class	401 - Residential
6	09-12-390-010	Gov Unit	09 - TEXAS TWP
7	09-12-390-010	Previous Class	401 - Residential
8	09-12-390-010	School	39140 - Portage Public
9	09-12-390-010	Owners Mailing Address	9999 BAY RIDGE RD,KALAMAZOO, MI 49009-8984
10	09-12-390-010	SEV 2009	83,700
11	09-12-390-010	Taxable 2009	83,700
12	09-12-390-010	SEV 2008	86,900
13	09-12-390-010	Taxable 2008	86,900
14	09-12-390-010	PRE	100%
15	09-12-390-010	Acreage	0
16	09-12-390-010	Legal Description	BAY RIDGE LOT 1* **9-94 1994 SPLIT FROM 12-326-020*
17	09-12-390-020	Parcel Number	09-12-390-020

Figure 5
Tax Information for Bay Ridge

The data can then be transposed up while converting a few columns from character to numeric.

```
proc transpose data=WORK.taxInfo out=WORK.taxInfo(drop=_:);
  by parcelNo notsorted;
  id label;
  var text;
run;

data WORK.taxInfo(drop=_:);
  set taxInfo (rename=(sev_2008=_SEV_2008 sev_2009=_SEV_2009));
  SEV_2008 = input(compress(_sev_2008, ' ,'), ?? best.);
  SEV_2009 = input(compress(_sev_2009, ' ,'), ?? best.);
run;
```

This returns a data set with one record per parcel.

PERFORMING A FEW SIMPLE ANALYTICS

The analytics for this example consist of simple rankings: actual ranking and percentile ranking. Further analysis could include such statistics as the year over year increase and the ratio of taxable value to actual value.

```
proc rank data=WORK.taxInfo out=WORK.taxInfo ties=mean;
   var sev_2009;
   ranks priceRank;
run;

proc rank data=WORK.taxInfo out=WORK.taxInfo ties=low groups=100;
   var sev_2009;
   ranks pricePct;
run;

proc sort data=WORK.taxInfo;
   by owners_name;
run;
```

CREATING A KEYHOLE MARKUP LANGUAGE (KML) DOCUMENT

The final step in our example is to generate the actual KML. KML is a flavor of XML which is used by Google Earth to add value (e.g. pushpins and annotations) to Google Earth.

- We could use Yahoo! web services to retrieve the latitude and longitude but in this case we will just let Google Earth implicitly handle the geocoding.
- We could also use the bidirectional XML access method in SAS 9.2 to generate the KML, but in this case we will instead use DATA NULLs; the XML access method would have wrapped the descriptions with CDATA tags and thus rendered them as straight text instead of as HTML.

The following code creates the three elements which will be sent to KML: name, description (with HTML) and address.

```
data WORK.taxInfo(drop=_:);
   if (_n_=1) then do;
      _prxName = prxparse('s/\s*&\s*/ and /i');
      retain _prx:;
   end;
   set WORK.taxInfo nobs=nobs;

   *** The name ***;
   length name $64;
   retain name '';

   *** The description ***;
   length description $256;
   description = catx('&lt;br&gt;',
      prxchange(_prxName, -1, owners_name),
      '&lt;hr&gt;',
      catx(' : ', '2009 SEV', put(sev_2009, dollar8.)),
      catx(' ', catx(' of ', put(priceRank, best.), put(nobs, best.)),
        cats('(', put(pricePct, best.), '%)')),
      '&lt;hr&gt;'
      );

   *** The address ***;
   length address $256;
   address = catx(',', property_address, '49009');
run;
```

The final step is to actually generate the KML document. The “Camera” element specifies the initial camera placement.

```
filename kmlOut "C:\Projects\neighborhood.kml";

data _null_;
   file kmlOut;
   put @1 '<?xml version="1.0" encoding="windows-1252" ?>';
   put @1 '<kml xmlns="http://www.opengis.net/kml/2.2">';
   put @3 '<Document>';
   put @5 '<Camera>';
   put @7 '<longitude>-85.659944</longitude>';
```

```
    put @7 '<latitude>42.218317</latitude>';
    put @7 '<altitude>1000</altitude>';
    put @7 '<altitudeMode>relativeToGround</altitudeMode>';
    put @5 '</Camera>';
    put @5 '<Folder>';
    STOP;
run;

data _null_;
    file kmlOut mod;
    set WORK.taxInfo;
    put @7 '<Placemark>';
    put @9 '<name>' name +(-1) '</name>';
    put @9 '<description>' description +(-1) '</description>';
    put @9 '<address>' address +(-1) '</address>';
    put @7 '</Placemark>';
run;

data _null_;
    file kmlOut mod;
    put @5 '</Folder>';
    put @3 '</Document>';
    put @1 '</kml>';
    STOP;
run;

filename kmlOut;
```

Once Google Earth has been installed on your machine, all that remains is to double click “neighborhood.kml” to display the results of our mashup (See Figure 6).

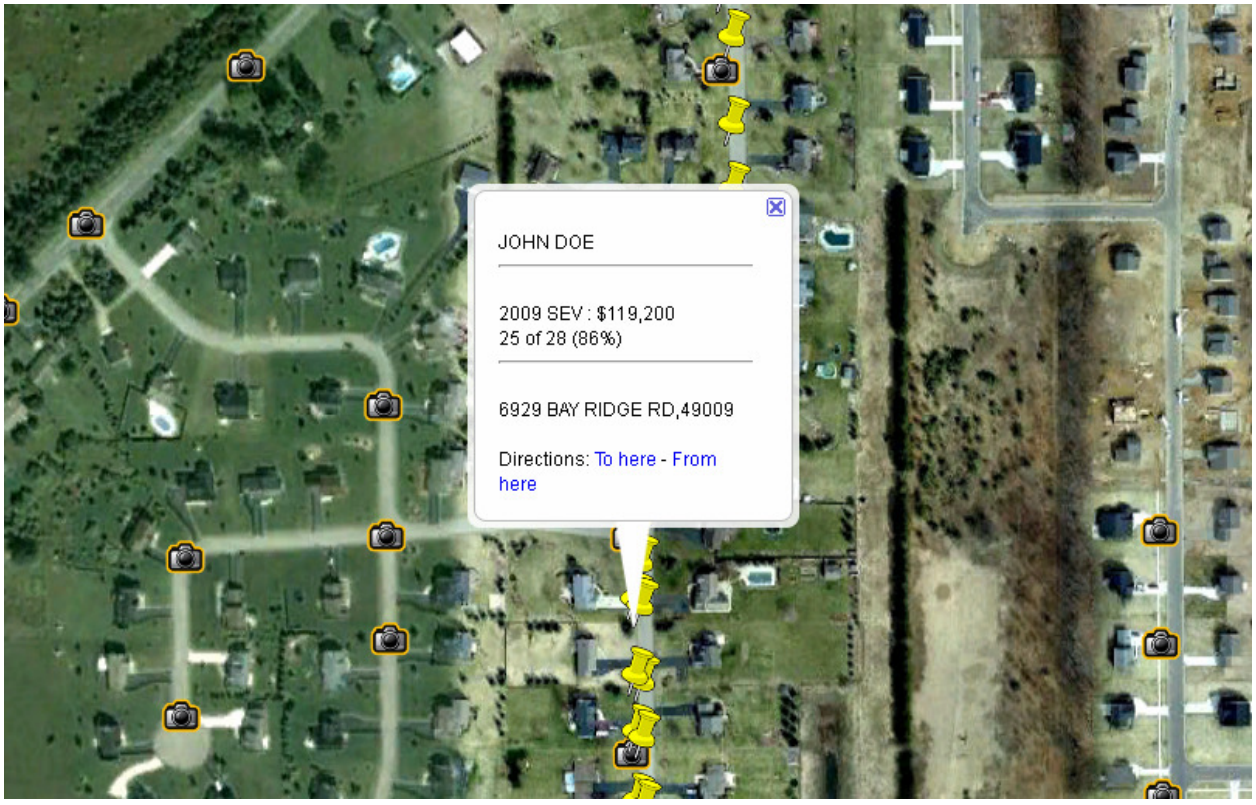


Figure 6
Using Google Earth to View Tax Information for the Neighborhood

CONCLUSION

I hope this paper has encourages you to think about SAS in a different way. SAS can be used to not only bring information to the Web; it can also be used to add value to information that is already on the Web. All that is needed is a little imagination.

CONTACT INFORMATION

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

Name:	Jack Fuller
Enterprise:	COMSYS Services LLC
Address:	5220 Lovers Lane
City, State ZIP:	Portage, MI 49002
Work Phone:	269.553.5126
Fax:	269.553.5198
E-mail:	jfuller@comsys.com
Web:	www.comsys.com