Paper RIV-02

A PICTURE is Worth Alot of PUTS

Carol Martell, Highway Safety Research Center, Chapel Hill, NC

ABSTRACT

This paper demonstrates the use of PICTURE formats to deliver SAS® data to nonstandard destinations, including OpenGIS® KML for Google Earth™ and ArcGIS®.

INTRODUCTION

I found myself needing a workaround for writing some SAS data as KML markup. In so doing, I came upon the PICTURE format statement, something I've ignored for many years. It simplified a PUT statement for that situation, so I looked around for other ways it might prove useful. Although data seems perfectly at home and well placed in a SAS table, I'm often required to deliver information in other modes. This paper makes some forays into incorporating the PICTURE format into the data output part of my little world. This paper is in no way an instruction manual in the use of the PICTURE format, but rather shows how the PICTURE format can move string constants from PUT statements to PICTURE statements.

DESTINATION: URL STRING

THE GOAL

A project requires a dynamic URL string containing a name/value pair that delivers the FIPS county code and a crash count for all 100 counties in North Carolina. Here's what it should look like:

```
fips1|count1;fips2|count2;fips3|count3;…fips99|count99;fips100|count100;
```

The output we need to write, reflecting the values from one table row, or one county, should look like this:

```
fips1|count1;
```

SUCCESS WITHOUT PICTURE FORMAT

This PUT statement, using the trailing '@', writes the output from all the data rows to a single line, but leaves spaces between the values – a problem in a URL string:

```
PUT fips  '|' count  ';' @;
```

Here is the output from the above PUT statement, with embedded blanks:

```
1 |100 ;3 |101 ;5 |104 ;7 |109 ;...
```

Adding pointer directives to the PUT statement removes the spaces and gives us what we need. We'll use this PUT statement for comparison with whatever solution we find using PICTURE formats:

```
PUT fips  +(-1) '|' count +(-1) ';' @;
```

Here is the output from the above PUT statement, without embedded blanks:

```
1|100;3|101;5|104;7|109;...
```

FINDING OUR WAY

We investigate what it takes to create the same output with PICTURE formats. We need two PICTURE formats – one for the FIPS code, and one for the count. We know that in North Carolina the FIPS codes range from 1-199. We do not, however, know the range of values for crash counts. Why does the range matter? Read any paper about PICTURE formats and it will say *KNOW YOUR DATA*! Why? You must know your data because PICTURE formats behave like a picture frame. For example, take a picture of three pumpkins in a whole field of pumpkins. Show the

picture to people, and all they see is three pumpkins – they don't know about the rest of the pumpkins. With a PICTURE format, it's like making a reservation at a very forgiving yet inflexible restaurant – you can make a reservation for 20, show up with 3 and they won't mind. However, if you show up with 21, they won't be dragging another table over, or even dragging up another chair. They won't complain – they will just completely ignore diner number 21. If you don't make a big enough 'reservation' for your data in the PICTURE format, anything extra is not displayed and there will be no ERROR message!

Let's isolate the formatting we need for the FIPS code. We want to write the FIPS code followed by the '|' character. Our output should look like this:

```
1|3|5|7|9|,,,197|199|
```

We'll try a PICTURE with three digits, since the maximum value is 199. We'll specify no leading zeros in our digit selectors, but if there are no crashes, we'd like to see the 0:

```
PICTURE x LOW-HIGH='009|';
```

We use this PUT statement to see what we get:

```
PUT fips x. @;
```

Unfortunately, it yields something with embedded blanks - a problem in a URL string:

```
 1|  3|  5|  7|  9| 11| 13| 15| 17| 19| 21| 23| 25| 27| 29| 31| 33|
```

We change the PICTURE to specify leading zeros:

```
PICTURE x LOW-HIGH='999|';
```

Using the same PUT statement, we have no more embedded blanks:

```
001|003|005|007|009|011|013|015|017|  019|021|023|025|027|029|031|033|
```

We could stop now, but suppose we really object to the embedded blanks. We try using ranges that supply different PICTURE lengths based on the number of crashes:

```
PICTURE x
      0-9='9|'
      10-99='99|'
      100-199='999|';
```

What?! I am still defeated by embedded blanks because the default length of a format is the longest formatted value – in this case 3 digits plus the '|'. Here is the result using our multi-range PICTURE format:

```
 1|  3|  5|  7|  9| 11| 13| 15| 17| 19| 21| 23| 25| 27| 29| 31| 33| 35| 37|
```

I become a stubborn person. Using my multi-range PICTURE format, I control the output by overriding the default format length. I base the length on the number of crashes, remembering to include space for '|':

```
SELECT;
      WHEN (fips<10) PUT fips x2. @;
      WHEN (fips<100) PUT fips x3. @;
      OTHERWISE PUT fips x4. @;
END;
```

This gives me the results I want, but at a cost of extra effort both in the PICTURE and the PUT statements.

```
1|3|5|7|9|11|13|15|17|19|21|23|25|27|29|31|33|35|37|
```

Do I really need the 3-range PICTURE statement if I supply the format lengths in my PUT statements?

```
PICTURE x LOW-HIGH='999|';
```

Using the above format, I find that I do not need 3 ranges. I do get what I want, but it is still at the cost of introducing the SELECT group into my code. I learned something for future use (overriding the default length), but I am lazy and avoid extra coding at all costs.

```
1|3|5|7|9|11|13|15|17|19|21|23|25|27|29|31|33|35|37|
```

Since I don't really mind leading zeros, I choose the simplest combination of PICTURE and PUT:

```
PICTURE x LOW-HIGH='999|';
PUT fips x. @;
```

Now that we have a FIPS format, we need a format for the crash counts. Here's what I know about my crash counts: there's no such thing a half a crash - you either have a crash or you don't, and you can't have fewer than no crashes. Therefore, while the values themselves are not known, I do know they will be integer and non-negative. These are PICTURE luxuries – no fractions to deal with, and no minus signs or parentheses to display negative numbers. Given the lessons learned working with the FIPS code, we begin by surrendering to the leading zero. We need a PICTURE format large enough to display the largest data value. We do not know what that maximum value is, so we do not know how many digit selectors to use. We would like to write code flexible enough to accommodate growing data sources. We insert a snippet of code to learn the maximum value of the data at hand, and measure that value to determine the length for the PICTURE format. In SQL, we find the number of digits in the MAX of our crash counts and store it into a macro variable. Then, we convert the MAX value itself into a character string with a PUT function that uses the macro-stored length. The resulting character string becomes the digit selector portion of our PICTURE format, and we follow it with a semicolon for our URL specs. We do not need to convert the digits to all 9's because any non-zero digit will suffice. Since 16 is the maximum number of digit selectors allowed, we use the format '16.' In a PUT function to convert the maximum value to the character string we measure to find the digit selector length:

```
PROC SQL NOPRINT;
      SELECT LENGTH(STRIP(PUT(MAX(crashcount),16.))) INTO :lngth FROM crashes;
      SELECT PUT(MAX(crashcount),%TRIM(&lngth..)) INTO :maxcount FROM crashes;
QUIT;
```

We now have information we can use for a dynamic PICTURE format:

```
PICTURE y LOW-HIGH="&maxcount;";
```

We use our new format with the crash counts in a PUT statement:

```
PUT count y. @;
```

Happily, our results are what we need:

```
0100;0101;0104;0109;0116;0125;0136;0149;0164;0181;
```

SUCCESS WITH PICTURE FORMAT

Now we can use the formats together to write our URL string!

```
PICTURE x LOW-HIGH='999|';
PICTURE y LOW-HIGH="&maxcount;";

PUT fips x. count y. @;

001|0100;003|0101;005|0104;007|0109;…197|9704;199|9901;
```

Comparing the above PUT statement with the original PUT statement we see that we've saved ourselves a little bit of PUT statement real estate, and avoided some danger from mismatched quotes.

```
OLD:   PUT fips  +(-1) '|' count +(-1) ';' @;
NEW:  PUT fips x. count y. @;
```

DESTINATION KML

THE GOAL

Given a SAS table with geographic coordinates, an numeric identifier, and a descriptor for a number of crashes, we'd like to create a KML document displaying the crashes to view in Google Earth. The document should have a placemark for each crash, labeled with the identifier, and should display the descriptor in the balloon. Each placemark corresponds to one row of data in our SAS table.

We examine what the barebones KML file should look like. The name tags surround text that becomes the label in Google Earth, and the description tags surround text that is displayed in the balloon for the placemark. The coordinates tags surround longitude, latitude, and altitude, and must not have embedded blanks. The opening two lines are required and should not be preceeded by blanks. The document tag is required if one wishes to display more than one placemark in the file. Here is our KML goal:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
      <Placemark>(one crash)
        <name>200700001(identifier)</name>
        <description>1600 block of Elm St(descriptor)</description>
        <Point>
          <coordinates>-122.0822035425683,37.42228990140251,0(coordinates)</coordinates>
        </Point>
      </Placemark>
      <Placemark>(next crash)
                ….
      </Placemark>
      <Placemark>
                ….
      </Placemark>
</Document>
</kml>
```

We will use the following table containing three rows of data:

crashid	description	longitude	latitude	altitude
200700001	100 block of Elm Street	-79.076298	35.912272	0
200700002	200 block of Elm Street	-79.076292	35.91353	0
200700003	200 block of E Poplar A	-79.076929	35.913061	0

SUCCESS WITHOUT PICTURE FORMAT

Here we see a more traditional PUT statement that intersperses data with markup tags. Data items and strings appear together in the PUT statements. We incorporate the means to write the beginning lines and the document closing tag by using the _n_=1 condition, and the END= option of the SET statement.

```
IF _n_=1 THEN
      PUT '<?xml version="1.0" encoding="UTF-8"?>'  /
      '<kml xmlns="http://www.opengis.net/kml/2.2">'  /
      '<Document>';

PUT '<Placemark><name>' crashid '</name><description>' description
      '</description><Point><Placemark><coordinates>' longitude +(-1) ',' latitude
      +(-1) ',' altitude '</coordinates></Point></Placemark>';

IF endflag THEN
PUT '</Document></kml>';
```

We use the above PUT statements, writing to an external file . Here is the resulting file. Please note that the lines have wrapped for display. The indented text is actually part of the line above in this text:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<Placemark><name>200700001 </name><description>100 block of Elm Street
      </description><Point><coordinates>-79.076298,35.912272,0 </coordinates></Point></Placemark>
<Placemark><name>200700002 </name><description>200 block of Elm Street
      </description><Point><coordinates>-79.076292,35.91353,0 </coordinates></Point></Placemark>
<Placemark><name>200700003 </name><description>200 block of E Poplar A
      </description><Point><coordinates>-79.076929,35.913061,0 </coordinates></Point></Placemark>
</Document></kml>
```

Viewing the file in Google Earth (**Figure 1**) we see the placemarks are labeled with the crash identifier. The open balloon displays the description. Both the identifier and description appear in the navigation pane to the left.

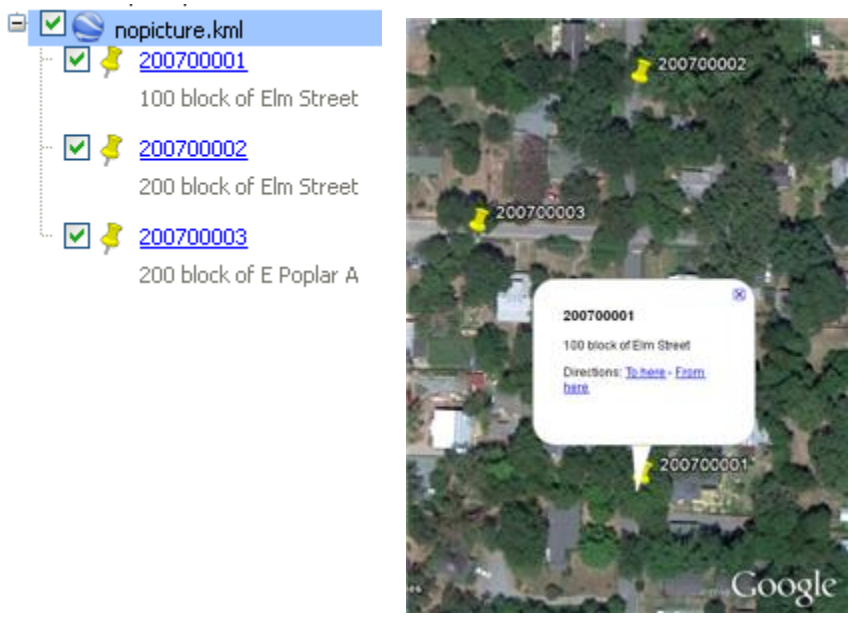

Figure 1. The KML file viewed in Google Earth.

SUCCESS WITH PICTURE FORMAT

To create the same output using PICTURE formats, we build a separate PICTURE format for each numeric variable. We surround the values with the KML tags using combinations of the PREFIX option and suffix strings.

The format for the numeric crash identifier will surround the value with <name> tags and become the map label. Our identifiers are always 9 digits long and are never negative or missing. The PREFIX opens the <Placemark> and <name> tags. After our digit selectors, we close the <name> tag and open the <description> tag:

```
PICTURE crid
    0-HIGH='123456789</name><description>' (PREFIX='<Placemark><name>');
```

The description field is character and will be written without a format.

The coordinates must appear as follows: the <coordinate> opening tag is followed by longitude. Longitude is followed by a comma and the latitude. Latitude is followed by a comma and the altitude. After altitude, we need the closing coordinate tag. There should be no embedded blanks in the numeric coordinate string.

For the longitude format, we use </description><Point><coordinate> as the PREFIX. This closes the <description> tag, and opens the <point> and <coordinates> tags. For values in the negative range, we must also add the minus sign. We place the comma after our digit selectors (‘999.999999,’) in the PICTURE statement. Alternatively, we could have incorporated that comma into the PREFIX for latitude.

```
PICTURE lng
    LOW-<0= '999.999999,' (prefix='</description><Point><coordinates>-')
    0-HIGH= '999.999999,' (prefix='</description><Point><coordinates>');
```

We build our latitude and altitude formats accordingly, using prefixes and/or inserting needed characters into the digit selector string. We avoid embedded blanks by using non-zero digit selectors, forcing the coordinates to the maximum length with zeros if need be. The altitude (never negative) PICTURE will close the coordinates, point and placemark tags.

```
PICTURE lat
    LOW-<0= '999.999999,' (prefix='-')
    0-HIGH= '999.999999,' ;
PICTURE alt
    0-HIGH= ‘999</coordinates></Point></Placemark>';
```

We use all four PICTURE formats in a single PUT statement to write the entire set of tags for a placemark:

```
PUT crashid crid. description longitude lng. latitude lat. altitude alt.;
```

The result is exactly what we need (the single line of output text is wrapped for display here):

```
<Placemark><name>200700001</name><description>100 block of Elm Street
    </description><Point><coordinates>-079.076298,035.912272,000</coordinates></Point></Placemark>
```

Now for our opening lines. We could write them using traditional PUT statements, or stubbornly insist on using PICTURE formats. PICTURE formats can only be used with numeric data. We do not, however, have to actually display the numeric data! We can use an existing numeric variable without displaying it by creating a PICTURE format having no digit selectors. We use ‘(NOEDIT)’ to prevent SAS from using numbers in the string as digit selectors. We cannot combine these into a single PICTURE format, because each must be written to a separate line:

```
PICTURE lineone LOW-HIGH='<?xml version="1.0" encoding="UTF-8"?>' (NOEDIT);
PICTURE linetwo LOW-HIGH='<kml xmlns="http://www.opengis.net/kml/2.2">' (NOEDIT);
PICTURE opendoc LOW-HIGH='<Document>' ;
```

Our numeric variable crashid is available to be invisible with our opening PICTURE formats:

```
IF _n_=1 THEN
      PUT crashid lineone. / crashid linetwo. / crashid opendoc.;
```

The result is shown here. Happily, the variable crashid is not displayed, and the text we need appears!

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
```

Likewise, we write the closing </Document> and </kml> tags, again using the numeric crashid for non-display. Here are the format, the PUT statement, and the resulting output:

```
PICTURE closedoc LOW-HIGH='</Document></kml>';

IF endflag THEN
      PUT crashid closedoc.;

</Document></kml>
```

Comparing the above PUT statements with the original PUT statements we see that this time we've saved quite a bit of real estate, again avoiding mismatched quote mishaps.

OLD:

```
 IF _n_=1 THEN
      PUT '<?xml version="1.0" encoding="UTF-8"?>' /
      '<kml xmlns="http://www.opengis.net/kml/2.2">' /
      '<Document>';
PUT '<Placemark><name>' crashid '</name>' '<description>' description
      '</description><Point><Placemark><coordinates>' longitude +(-1) ',' latitude
      +(-1) ',' altitude '</coordinates></Point></Placemark>';
IF endflag THEN
      PUT '</Document></kml>';
```

NEW:

```
IF _n_=1 THEN
      PUT crashid lineone. / crashid linetwo. / crashid opendoc.;
PUT crashid crid. description longitude lng. latitude lat. altitude alt.;
IF endflag THEN
      PUT crashid closedoc.;
```

CONCLUSION

Through exploration with our scenarios, we've learned a few things:

1) PICTURE formats can simplify PUT statements. In addition to shortening the statement, PICTURE formats remove the static string from the PUT statement, making it visually less confusing. Thankfully, if the same information is to be written repeatedly, the programmer need only match those pesky quotes one time in the FORMAT procedure. There could be some advantage if the need arose to change the embedded characters. For instance, if the required delimiters for the URL string example were to change, I'd only need to change the PICTURE statement rather than find and replaces instances in PUT statements.
2) Creating multiple PICTURE ranges and/or overriding default format lengths in the PUT statement provides more granular control of the output.
3) We took a peek at creating a dynamic PICTURE format. In our example, the number of digit selectors change to fit the current data. The suffix and PREFIX could also become dynamic.
4) We have seen that a character string constant can be output using a PICTURE format. Any numeric field can stand in for a PICTURE that has no digit selectors. Again, there could be some advantage to having the string(s) located in the FORMAT procedure rather than in PUT statements if changes are required.

RECOMMENDED READING

PICTURE formats are confusing and not intuitive. Once you understand them, they simply become exacting. The SAS documentation is always recommended reading. I read several papers that helped me better understand the topic: Pete Lund's "More than Just Value: A Look Into the Depths of PROC FORMAT", Carry Croghan's "PICTURE Perfect: In depth look at the PICTURE format", and Andrew Karp's "Getting in to the Picture (Format)".

CONTACT INFORMATION

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

Carol Martell
Highway Safety Research Center
730 Martin Luther King Jr. Blvd.
Chapel Hill, NC 27514
Work Phone: 919-962-8713
E-mail: carol_martell@unc.edu