Paper 069-2009

"Excel"lent SAS® Formulas: The Creation and Export of Excel Formulas Using SAS

Collin Elliot, Itron, Inc., Vancouver, WA
Eli Morris, The Cadmus Group, Portland, OR

ABSTRACT

In cases where there is a need to trace and understand the underlying inputs and calculations, Microsoft Excel is the industry standard for presenting data. While it is common practice to export SAS data to Excel for presentation and review, this does not provide the desired transparency because the data values do not contain any of the linked formulas or calculations. The most obvious solution to this problem would be to export the necessary data to Excel and build the calculated fields manually. However, this is not practical when working with large data sets, complex formulas, and/or data that change frequently. Our approach consists of creating the formulas as text strings in SAS and exporting them to Excel so that they function as desired with no additional user intervention. This approach establishes correct Excel address references based on SAS variable names, and can be used to reference single cells, ranges, named ranges, previous rows, subsequent rows, or any other type of address that might be required - all without a single keystroke in Excel.

INTRODUCTION

Throughout our careers we have frequently found ourselves dealing with clients and colleagues who expect to see data in Excel, even when the analysis has been performed in SAS. To further complicate matters, it is often deemed insufficient to merely present the numbers without the formulas underlying the calculations. Our initial attempts to satisfy these requests consisted of simply exporting the data into Excel and creating the formulas manually. It was a sufficient one-time solution, but as the frequency and complexity of these requests increased, it became clear that we needed to develop an automated solution. One option was to use Visual Basic for Applications (VBA) within Excel, but this merely introduced another step in the process and was often not in the skill-set of the SAS programmer performing the analysis. Therefore, our goal was to solve the problem in SAS, minimizing the amount of effort necessary between running the SAS code and reviewing the Excel workbook. After much trial and error, we devised a system to accomplish this task by building text strings that, once exported into Excel, act as the required formulas. This paper explains our approach and shows the steps required for its execution.

A SIMPLE EXAMPLE

The following example provides a basic illustration of the concept. Imagine you work for an electric utility that runs a program to encourage its customers to install energy-efficient compact fluorescent light bulbs (CFLs) in their homes. Program staff members have collected data with the number of CFLs installed and the total kilowatt-hour (kWh) savings associated with those installations. The program manager wants you to provide an Excel workbook so that she can assess the kWh savings per CFL. While the computation is simple enough, the program manager wants to see the calculation as a formula in Excel. Given this requirement, instead of exporting the data and creating the necessary formula in Excel by hand, you decide to create the formula as a text string in SAS and then export the data to Excel. The following steps demonstrate this task:

1. Create a single observation of total savings, installations, and a text string in SAS that will represent the correct formula upon export, as shown in Figure 1, so that the calculation takes place in Excel. Note that the fourth row is used here because of the presence of a title. Without a title, the data would appear in the second row and the bulbSavings would use "=A2/B2" for the formula.

```
data savings;
    kWhSavings = 9400;
    installs = 143;
    bulbSavings= "=A4/B4";
run;

proc print data = savings noobs;
    title 'Per-Bulb Savings';
run;
```

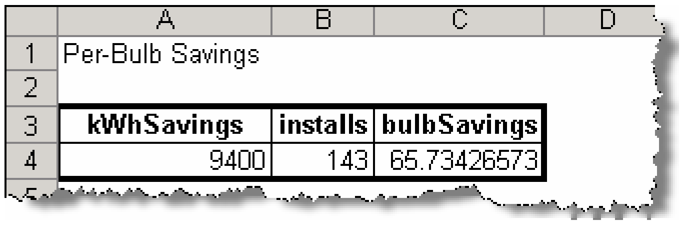
```
             Per-Bulb Savings

      kWh                           bulb
    Savings      installs         Savings

      9400          143           =A4/B4
```

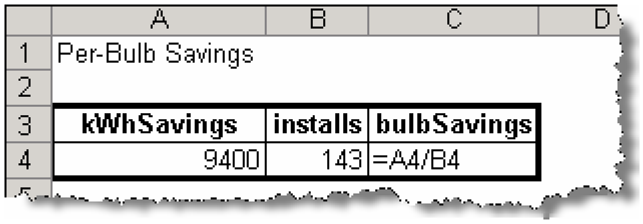
Figure 1. Example of Text String Formula and SAS Listing Output

2. With the text string created to calculate the per-bulb savings, you can send the data to Excel by using the "View in Excel" option available in the SAS Explorer, which exports the data using ODS HTML. Figure 2 shows how the data appear in Excel upon export and Figure 3 shows the underlying formula.



	A	B	C	D
1	Per-Bulb Savings			
2				
3	kWhSavings	installs	bulbSavings	
4	9400	143	65.73426573	

Figure 2. Exported Data with Formula Resolved



	A	B	C	D
1	Per-Bulb Savings			
2				
3	kWhSavings	installs	bulbSavings	
4	9400	143	=A4/B4	

Figure 3. Exported Data Showing Formula

This example only requires a single formula and consists of one observation of data, so while it illustrates the general idea, it fails to show why the creation and export of text string formulas presents a problem that calls for an automated solution. There are many cases, however, where an automated solution is strongly preferable, mainly due to:

- **Quantity**: The tables that need to be exported to Excel are so numerous that it would be too time consuming to manually create the formulas, no matter how simple the calculation.
- **Complexity**: The Excel spreadsheets contain multiple columns with formulas and/or complex formulas, particularly those that cannot simply be created once they are copied or auto-filled.
- **Replicability**: The original data are subject to frequent update, meaning that any effort made to create the formulaic columns in Excel might have to be performed repeatedly.

In cases meeting at least one of the above criteria, the task of generating the text string formulas will likely be far more complicated than the example illustrated above. Just the task of figuring out which columns and rows correspond to different variables and observations in SAS will be much more difficult. Even if one overcomes this obstacle through brute force programming, the resulting code would be onerous to interpret and even more difficult to modify. Our approach is an attempt to overcome these difficulties to allow the analyst to focus on the calculations that need to be created, not on the locations of the necessary variables.

STEPS TO THE APPROACH

The five main steps associated with our approach are:

1. Create a template data set in SAS that represents what the data—both actual data values and formulas—will look like in Excel.
2. Based on the template data set, use the FORMAT procedure to create an INFORMAT to match each variable to its corresponding Excel column (A, B, etc.).
3. Using the INFORMAT from the previous step, create a series of arrays in a DATA STEP that use alias variable names from the template, but contains text values for the Excel address of the different columns (A1, A2, etc.).
4. Referring to the variables aliases in the arrays, create the text string formulas that will appear in Excel.
5. With the text string formulas complete, export the data to Excel using the most appropriate method for the particular situation.

Each of these steps has many underlying details. The best way to illustrate them is by providing a step-by-step example of the actual implementation of our approach, which we will accomplish by expanding on our original example.

A MORE COMPLICATED STEP-BY-STEP EXAMPLE

Returning to the example of the utility energy efficiency program, staff members have built a database to store information on the energy-efficiency measures that customers have installed. Figure 4 shows the code to create a small subset of these data along with the SAS LISTING output from the PRINT procedure.

```
data programData;
    length business $7 measure $18;
    infile datalines dlm = ',';
    input business $ measure $
installs kWhSavings;
    label business = 'Business Type'
          measure = 'Measure'
          installs = 'Installations'
          kWhSavings = 'kWh Savings';
datalines;
Grocery, T8, 406, 24365
Grocery, Strip Curtains, 32, 11985
Office, CFL, 97, 16510
Office, LED Exit Signs, 105, 35412
Office, T8, 41, 1811
Office, Occupancy Sensor, 493, 218976
Retail, CFL, 158, 27991
Retail, LED Exit Signs, 4, 1124
Retail, T8, 601, 35089
;
run;

proc print data = programData noobs;
    title 'Example Data';
run;
```

```
                      Example Data

                                                 kWh
business    measure             installs     Savings

Grocery     T8                      406        24365
Grocery     Strip Curtains           32        11985
Office      CFL                      97        16510
Office      LED Exit Signs          105        35412
Office      T8                       41         1811
Office      Occupancy Sensor        493       218976
Retail      CFL                     158        27991
Retail      LED Exit Signs            4         1124
Retail      T8                      601        35089
```

Figure 4. SAS Code to Create Sample Program Data and Listing Output

Based on the database, the program manager wants you to generate an Excel report that shows the following:

- The type of business (business)
- The energy efficiency measure (measure)
- The number of measures installed (installs)
- The total kWh savings associated with the installed measures (kWhSavings)
- The savings per installation (kWhPerMeasure)
- The cumulative savings in each business type (cumBizSavings)
- The measure savings as a percent of the business type's total savings (pctOfBiz)
- The measure savings associated with a given business type as a percent of the measure's total savings (pctOfMeasure)

While the first four variables are in the database, the remaining must be calculated. The program manager has explicitly stated that she wants to see the underlying calculations as formulas in Excel. Given the program manager's requirements, the following steps illustrate how you would apply our approach to accomplish this assignment.

STEP 1: CREATE A TEMPLATE DATA SET

The first step in our approach is to create a template SAS data set that contains the variables that already exist as well as placeholders for the variables that will contain the Excel formulas. This step requires planning what the Excel workbook will look like—which columns will be data, which will be formulas—and creating a version of it in SAS. While the location of a variable rarely matters in SAS, Excel relies on the position of data in the workbook to perform calculations, **so the order of the variables in the template is crucial and must remain constant**. Our approach creates this template using The SQL procedure, which provides more transparent and explicit control of the order and attributes of the variables, although this could also be accomplished through a DATA STEP.

In our example, we have four variables (business, measure, installs, and kWhSavings) from the database and four that will be formulas. The PROC SQL code to create the data template is shown in Figure 5.

```
proc sql;
    CREATE TABLE dataTemplate AS
    SELECT business 'Business',
           measure 'Measure',
           installs 'Installs',
           kWhSavings 'Total kWh Savings',
           ' ' AS kWhPerMeasure length 100 'kWh Savings per Measure'
           ' ' AS cumBizSavings length 100 'Cumulative kWh Savings for Business',
           ' ' AS pctOfBiz length 100 'kWh Savings as % of Business',
           ' ' AS pctOfMeasure length 100  'kWh Savings as % of Measure'
    FROM programData
    ORDER BY business, measure;
quit;
```

Figure 5. SAS Code to Create Data Template for Step 1

There are three aspects of the code in Figure 5 that require discussion. The first is that the order of the variables needs to be exactly as they will appear in Excel. If one wanted the savings per measure to come last, it would need to be last in the query. The second is that the placeholders for the three formula variables—even though they resolve to numeric values in Excel—need to be character variables, since they will eventually contain text strings. The third is that the formula placeholders should have a length sufficient to accommodate the formulas they will contain.

The data template will be the input to a number of subsequent steps, all of which will add additional variables and alter its structure. Because the final output of this approach must be identical in structure to the data template, it is helpful to query the SASHELP VCOLUMN view of this initial version of the data template. This creates a macro variable to use in a final PROC SQL query that will select—and in the correct order—only those variables from the data template. The syntax for this query is shown in Figure 6.

```
proc sql;
    SELECT name INTO :finalQuery
    SEPARATED BY ', '
    FROM sashelp.vcolumn
    WHERE libname = "WORK" AND
            memname = "DATATEMPLATE"
    ORDER BY varNum;
quit;
```

Figure 6. SAS Code to Create a Macro Variable for the Final Query

STEP 2: CREATE FORMATS TO MAP SAS VARIABLES TO EXCEL COLUMNS

Having established what the Excel workbook will look like by creating the data template, the next step is to associate each of the variables in this data set with its Excel column. There are two crucial components to this step. The first is to create a SAS FORMAT that will match positions in a generic SAS data set to Excel column names by mapping the numeric position of a variable to its corresponding Excel columns (e.g., 1 = “A”, 2 = “B” . . . 256 = “IV”). This format is created through a data set with the number to alphabet mapping that acts as the CNTLIN data set in PROC FORMAT, which is shown in Figure 7. This FORMAT will be identical every time one applies this approach, so depending on how often the approach is employed, it could be created once and saved in a permanent format catalog.

```
data excelColumns(where = (start <= 256));
    length label $2;
    do c1 = -1 to 25;
        do c2 = 0 to 25;
            alpha1 = byte(rank('A')+ c1);
                  alpha2 = byte(rank('A')+ c2);
            label = compress(alpha1||alpha2, " @");
            start + 1;
            fmtName = "column";
            output;
        end;
    end;
run;

proc format cntlin = excelColumns;
run;
```

Figure 7. SAS Code to Create a FORMAT that Maps Variable Positions to Excel Columns

The second component of this step creates a CNTLIN data set from the SASHELP VCOLUMN view, which has the position of each variable in the data set created in Step 1. The Excel column FORMAT associates each SAS variable with its Excel column name. Once this matching is complete, an INFORMAT that will convert variable names to Excel columns is created for later use. The code to create the INFORMAT is presented in Figure 8 and the CNTLIN data set is shown in Figure 9.

```
proc sql;
    CREATE TABLE addressFmt AS
    SELECT strip(upcase(name)) AS start,
           strip(put(varnum, column.)) AS label length 2,
           "var2Excel" AS fmtName,
           "J" AS type,
           type AS varType,
           varNum
    FROM sashelp.vcolumn
    WHERE libname = "WORK" AND
            memname = "DATATEMPLATE"
    ORDER BY varNum;
quit;

proc format cntlin = addressFmt;
run;

proc print data = addressFmt;
     title 'Address Format';
run;
```

Figure 8. SAS Code to Create an INFORMAT that Maps Variable Names to Excel Columns

```
          Address Format

Obs    start            label     fmtName     type

1      BUSINESS           A      var2Excel     J
2      MEASURE            B      var2Excel     J
3      INSTALLS           C      var2Excel     J
4      KWHSAVINGS         D      var2Excel     J
5      CUMBIZSAVINGS      E      var2Excel     J
6      PCTOFBIZ           F      var2Excel     J
7      PCTOFMEASURE       G      var2Excel     J
```

Figure 9. SAS Listing of the Sample Variable to Column INFORMAT

STEP 3: CREATE EXCEL ADDRESS ARRAYS

Once a relationship is established between the variables in the data template and their future locations in Excel, the next step is to create a series of character arrays that contain different types of Excel addresses that will be used to create the formulas. By different types of addresses, we are referring to whether an address represents a cell or a range and its relative position to the cell in which it appears. The most common address will refer to a different column in the current row, but there are many different types of possible address references. An address might refer to the previous row for either the same or another column, for example. Ranges are another type of address reference that will occur frequently. **This step requires considerable planning on the part of the programmer, since it is necessary to know what formulas will be in the Excel workbook and what type of cell references they will require.**

In our approach, the names of the variables in the address arrays will be based on the names of the variables in the data template, but with an added suffix to indicate what type of address the array contains. The suffixes used for the arrays are up to the programmer, but it is helpful to use consistent and, hopefully, intuitive conventions in selecting them. For this example each type of address reference and its associated suffix is presented in Figure 10. Note that these are merely the addresses necessary for this example. Additional arrays could easily be created for other address types you might need.

Suffix	Description
_X	Address for the Current Row: A[Current Row]
_G	Address for the first row of each group for a grouping variable: A[First Row in Group]
_R	A range that goes from the first row of the group to the current row: A[First Row in Group]:A[Current Row].
_C	A range that goes from the first to the last row in the data set: A[First Row]:A[Last Row]

Figure 10. Excel Address Types Used in Example

To build these arrays, we use PROC SQL to query the CNTLIN data set from Step 2 to create macro variables of the variables in each array reference. The code used to generate the array variable lists is presented in Figure 11.

```
%macro arrayMacros(arrayType);

    %global excel&arrayType;

    proc sql noprint;
        SELECT strip(start)||"&arrayType" INTO :excel&arrayType
        SEPARATED BY " "
        FROM addressFmt
        ORDER BY varnum;
    quit;

%mend arrayMacros;
%arrayMacros(_X);
%arrayMacros(_G);
%arrayMacros(_R);
%arrayMacros(_C);
```

Figure 11. SAS Code to Create Macro Variables for Address Array References

The macro variables for the array lists are used in a DATA STEP with an ARRAY statement to create the variables that will be populated with the Excel addresses. As shown in Figure 12, the address values are created by concatenation of the Excel column with the row number appropriate for the desired type of address. Excel range addresses are created by concatenating previously defined addresses. A key statement in the SAS code is the %LET statement that creates the "rowAdder" macro variable. The value of this will depend on where the first row of data will appear in the Excel spreadsheet, which depends on the method of export (discussed in Step 5). It is worth noting that this step could certainly be done through brute force programming, but the advantage of using our approach is that the macro variables are data-driven based on the data template. The code will dynamically adjust to any changes made to the number or order of the variables in the data template.

```
%let rowAdder = 3;

proc sql;
    SELECT strip(put(count(*) + &rowAdder, best.)) INTO :lastRow
    FROM dataTemplate;
quit;

data dataTemplate2;
    length &excel_X &excel_G &excel_R &excel_C $40;
    retain firstBizRow;
    set dataTemplate;
    by business measure;

    array addr_X {*} $ &excel_X;
    array addr_G {*} $ &excel_G;
    array addr_R {*} $ &excel_R;
    array addr_C {*} $ &excel_C;

    firstRow = strip(put(1 + &rowAdder, best.));
    lastRow = "&lastRow";
    currentRow = strip(put(_N_ + &rowAdder, best.));
    if first.business then firstBizRow = currentRow;

    do i = 1 to dim(addr_X);

         xlsName = upcase(scan(vname(addr_x(i)), 1, '_'));
         xlsColumn = strip(input(xlsName, $var2excel.));

         addr_x(i) = cats(xlsColumn, currentRow);
         addr_g(i) = cats(xlsColumn, firstBizRow);
         addr_r(i) = strip(addr_g(i))||":"||strip(addr_x(i));
         addr_c(i) = compress(cats(xlsColumn, firstRow, ':', xlsColumn, lastRow));

    end;

run;
```

Figure 12. SAS DATA STEP to Create Excel Address Arrays

The first three observations for the address arrays are shown in Figure 13. One of our goals in designing this approach was to allow the programmer to think about SAS variable names, and not Excel columns, when creating the formulas. The programmer need only know which suffix applies to the correct type of address to create the correct Excel reference. Note that by adding these suffixes to the original variable names, one will want to make sure that all of the original variable names are short enough to accommodate the additional characters.

```
                              Variables for the _X Array

BUSINESS_    MEASURE_    INSTALLS_    KWHSAVINGS_    CUMBIZSAVINGS_    PCTOFBIZ_    PCTOFMEASURE_
X           X           X             X                X              X               X

A4           B4          C4            D4               E4             F4              G4
A5           B5          C5            D5               E5             F5              G5
A6           B6          C6            D6               E6             F6              G6

                              Variables for the _G Array

BUSINESS_    MEASURE_    INSTALLS_    KWHSAVINGS_    CUMBIZSAVINGS_    PCTOFBIZ_    PCTOFMEASURE_
G           G           G             G                G              G               G

A4           B4          C4            D4               E4             F4              G4
A4           B4          C4            D4               E4             F4              G4
A6           B6          C6            D6               E6             F6              G6

                              Variables for the _R Array

BUSINESS_    MEASURE_    INSTALLS_    KWHSAVINGS_    CUMBIZSAVINGS_    PCTOFBIZ_    PCTOFMEASURE_
R           R           R             R                R              R               R

A4:A4       B4:B4       C4:C4         D4:D4            E4:E4          F4:F4           G4:G4
A4:A5       B4:B5       C4:C5         D4:D5            E4:E5          F4:F5           G4:G5
A6:A6       B6:B6       C6:C6         D6:D6            E6:E6          F6:F6           G6:G6

                              Variables for the _C Array

BUSINESS_    MEASURE_    INSTALLS_    KWHSAVINGS_    CUMBIZSAVINGS_    PCTOFBIZ_    PCTOFMEASURE_
C           C           C             C                C              C               C

A4:A12       B4:B12      C4:C12        D4:D12           E4:E12         F4:F12          G4:G12
A4:A12       B4:B12      C4:C12        D4:D12           E4:E12         F4:F12          G4:G12
A4:A12       B4:B12      C4:C12        D4:D12           E4:E12         F4:F12          G4:G12
```

Figure 13. SAS Listing Output of the Excel Address Arrays

STEP 4: BUILD THE EXCEL FORMULA TEXT STRINGS

The previous step created a data set with a series of arrays containing Excel addresses. To represent Excel formulas, these addresses need to be combined with an equal sign ("=") and any other characters ("*", "/", "-", etc.) necessary to produce the desired results. While it depends entirely on the specific requirements of the project, this step can be as simple as concatenating a set of addresses with the appropriate operator or as complicated as building complex formulas with multiple advanced Excel functions. The remainder of this section describes how to construct the Excel formulas for the four fields in our example using the Excel addresses developed in Step 3.

The variable kWhPerMeasure is the most straightforward of the four fields, calling only for the division of the kWhSavings variable by the installs variable in the same row. If we assume that our first observation in Excel is in the fourth row, for example, the formula should be "=D4/C4" because kWhSavings is in the fourth column and installs is in the third. In our approach, however, the programmer does not need to focus on location of the variables, but simply the calculations. To replicate this formula with the defined variables, the programmer need only refer to the data in the current observation (array _x), as shown in statement ① in Figure 14. Note that in constructing all formulas we rely on the CATS function to combine text strings because it conveniently strips away any unwanted blanks and results in more legible code.

```
data dataTemplate3;
    set dataTemplate2;

    ① kWhPerMeasure = cats('=', kWhSavings_X, ' / ', installs_x);

    ② cumBizSavings = cats('=sum(', kWhSavings_r, ')');

    ③ pctOfBiz = cats('=', kWhSavings_x, '/ sumif(', business_c, ',',
                         business_x, ',', kWhSavings_c, ')');

    ④ pctOfMeasure = cats('=', kWhSavings_x, '/ sumif(', measure_c, ',',
                              measure_x, ',', kWhSavings_c, ')');

run;
```

Figure 14. Creation of the Text String Formula for kWhPerMeasure

For cumBizSavings, the formula relies on an array that ranges from the first row for a given business type to the current row. In our sample data (Figure 4), the first two rows of data are within the building group “Grocery”. Suppose, given a row adder of 3, that we would like to calculate the cumulative savings of a measure on row five (the second measure in this group). The Excel formula would be “=sum(D4:D5)”. Note that this formula would not be easy to replicate manually for all rows in the workbook. However, based on the addresses created for the _R arrays, our approach allows a simple SAS statement to create the appropriate formula for all rows in Excel (statement ② in Figure 14).

The formulas for the pctOfBiz and pctOfMeasure variables require dividing the kWhSavings variable in a given row by the sum of the kWhSavings for a given business or measure. The easiest solution for this formula is to use the Excel SUMIF function, which has three arguments: comparison range, criteria, and sum range. Both the comparison and sum ranges require address ranges that extend from the first through the last row of data, so the _C array will work for both arguments. In our example, the nine observations will reside in rows four through twelve, so for the first row of data for the pctOfBiz variable, the formula would be “=D4/ SUMIF(A4:A12, A4, D4:D12)”. The construction of the text string formulas for both pctOfBiz and pctOfMeasure are presented in statements ③ and ④ in Figure 14.

Once the formula text strings have been constructed, we need to create a final data set that contains only the variables in the data template and in the correct order. Figure 15 shows the PROC SQL query to create this final data set, relying on the macro variable created in Figure 6.

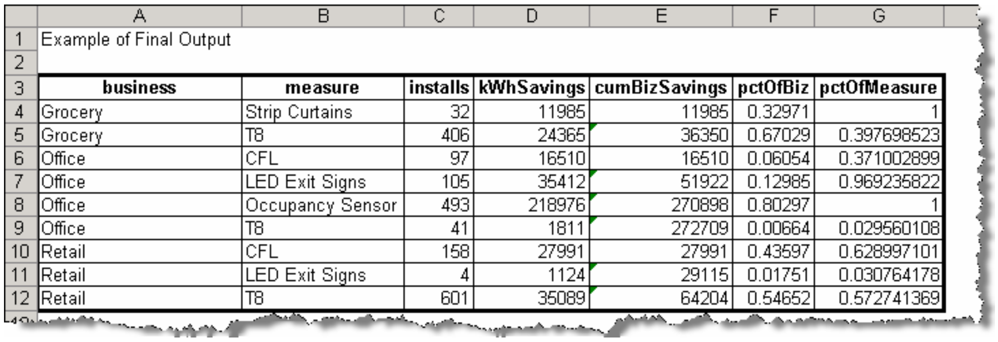
```
title 'Example of Final Output';

proc sql;
    CREATE TABLE finalData AS
    SELECT &finalQuery
    FROM dataTemplate3;
quit;
```

Figure 15. PROC SQL Query to Produce Final Data Set

STEP 5: EXPORT DATA TO EXCEL

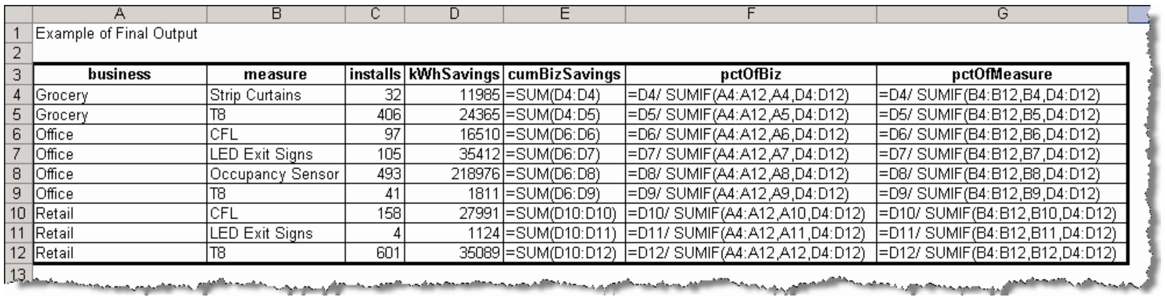
Once you have completed steps one through four, the task becomes exporting the final data set into Excel. The example in this paper has assumed you will simply use the “View in Excel” option through the SAS Explorer to export the data to Excel, which will produce the Excel output presented in Figure 16, also shown with the resolved formulas in Figure 17. With this method, the text string formulas resolve immediately in Excel.



Example of Final Output

business	measure	installs	kWhSavings	cumBizSavings	pctOfBiz	pctOfMeasure
Grocery	Strip Curtains	32	11985	11985	0.32971	1
Grocery	T8	406	24365	36350	0.67029	0.397698523
Office	CFL	97	16510	16510	0.06054	0.371002899
Office	LED Exit Signs	105	35412	51922	0.12985	0.969235822
Office	Occupancy Sensor	493	218976	270898	0.80297	1
Office	T8	41	1811	272709	0.00664	0.029560108
Retail	CFL	158	27991	27991	0.43597	0.628997101
Retail	LED Exit Signs	4	1124	29115	0.01751	0.030764178
Retail	T8	601	35089	64204	0.54652	0.572741369

Figure 16. Final Output Using SAS Explorer “View in Excel” Option, Formulas Resolved



Example of Final Output

business	measure	installs	kWhSavings	cumBizSavings	pctOfBiz	pctOfMeasure
Grocery	Strip Curtains	32	11985	=SUM(D4:D4)	=D4/ SUMIF(A4:A12,A4,D4:D12)	=D4/ SUMIF(B4:B12,B4,D4:D12)
Grocery	T8	406	24365	=SUM(D4:D5)	=D5/ SUMIF(A4:A12,A5,D4:D12)	=D5/ SUMIF(B4:B12,B5,D4:D12)
Office	CFL	97	16510	=SUM(D6:D6)	=D6/ SUMIF(A4:A12,A6,D4:D12)	=D6/ SUMIF(B4:B12,B6,D4:D12)
Office	LED Exit Signs	105	35412	=SUM(D6:D7)	=D7/ SUMIF(A4:A12,A7,D4:D12)	=D7/ SUMIF(B4:B12,B7,D4:D12)
Office	Occupancy Sensor	493	218976	=SUM(D6:D8)	=D8/ SUMIF(A4:A12,A8,D4:D12)	=D8/ SUMIF(B4:B12,B8,D4:D12)
Office	T8	41	1811	=SUM(D6:D9)	=D9/ SUMIF(A4:A12,A9,D4:D12)	=D9/ SUMIF(B4:B12,B9,D4:D12)
Retail	CFL	158	27991	=SUM(D10:D10)	=D10/ SUMIF(A4:A12,A10,D4:D12)	=D10/ SUMIF(B4:B12,B10,D4:D12)
Retail	LED Exit Signs	4	1124	=SUM(D10:D11)	=D11/ SUMIF(A4:A12,A11,D4:D12)	=D11/ SUMIF(B4:B12,B11,D4:D12)
Retail	T8	601	35089	=SUM(D10:D12)	=D12/ SUMIF(A4:A12,A12,D4:D12)	=D12/ SUMIF(B4:B12,B12,D4:D12)

Figure 17. Final Output Using SAS Explorer “View in Excel” Option, Formulas Shown

While the “View in Excel” demonstrates the successful export of the text string formulas, there are a number of potential problems with the output. For one, the data are not formatted for immediate presentation. Additionally, the output is a temporary Excel file that must be saved to a permanent directory, whose location will not be evident from looking at the SAS code used to generate it. Depending on one's needs, there is potentially a lot of manual work to be done before the product is final. Fortunately there are multiple methods for exporting data into Excel. The emphasis of this paper is not on the details of how to export the data, but there are a number of considerations of particular importance to the efficient export of this dataset:

- **Recognition of Text Strings:** The first consideration is whether Excel recognizes the exported text strings as formulas. Some methods (the EXPORT procedure and LIBNAME Excel) embed a special character before the text string so that the cells are treated as text and the formulas do not resolve. There are ways to remove these characters, but this is a significant disadvantage to using these methods.
- **Automatic Workbook Creation:** Most export methods will create the output workbook if it does not already exist. For those methods that do not create workbooks (Dynamic Data Exchange (DDE) is the main example), the user will need to create the workbook ahead of time, removing some of the automation from the process.
- **Pre-formatting:** Depending on the complexity and number of worksheets in a workbook, an analyst can spend a significant amount of time formatting the data for presentation. For efficient deployment, it is important that if data are updated, this formatting does not need to be redone manually. This goes hand-in-hand with the previous bullet, as most methods do not allow SAS to pre-format Excel workbooks. Cutting and pasting the data or formats is a potential solution, but this introduces one more step into the process.
- **Export of Variable Labels:** Because of SAS requirements on variable naming (no spaces, cannot begin with a number, etc.), SAS variable names are often less desirable for presentation purposes. To make the column headers more useful, the analyst can either manually change them in Excel or select a method of export that allows SAS variable labels to be used as Excel column headers.
- **Range Specification:** Most methods export data to Excel starting in cell A1 on the specified sheet. Some methods, however, allow users to specify alternative starting positions. This capability may prove useful in many instances.
- **Export of multiple datasets to the same worksheet:** Most export routines will send a SAS data set to its own individual sheet. However, in some applications, it may be useful to show multiple tables on the same sheet. For example, presenting both detailed data along with summary results.
- **Export of multiple worksheets to the same workbook:** Often, many tables will need to be created at once and the easiest way to organize and transfer the data is in a single Excel workbook. Some export methods will allow the user to automatically create one workbook with many sheets, while some will require the user to copy and paste manually.

Figure 18 below shows the considerations above cross-referenced with different export options. The list of export options is not intended to be exhaustive, but instead inclusive of the most common and effective means of export to Excel.

Consideration	"View in Excel"	PROC EXPORT	ODS CSV w/ PROC Print	ODS HTML w/ PROC Print	LIBNAME EXCEL	DDE	*ODS excelXP Tagset*
Automatic recognition of text strings as Excel formulas	✓		✓	✓		✓	✓
Automatic workbook creation	✓	✓	✓		✓		✓
Pre-formatting				✓		✓	✓
Ability to export to specific Excel ranges						✓	✓
Export of multiple datasets to the same worksheet						✓	✓
Exporting variable labels instead of names			✓	✓	✓	✓	✓
Export of multiple worksheets to the same workbook		✓			✓	✓	✓

Figure 18. Matrix of Export Methods and Capabilities

Given the matrix presented in Figure 18, the best method to export data would seem to be ODS excelXP tagset. However, this method comes with the significant caveat that the recognition of the text string formulas does not work unless the cell addresses are based on the "R1C1" (where "R1" represents the first row in Excel, counting from the top and "C1" represents the first column, counting form the left) reference style, as opposed to the "A1" style. Fortunately, the code only requires two modifications to use this approach.

The first change is in Step 2, where the INFORMAT is created with PROC FORMAT to map the variables names to the Excel columns. The conversion from numeric position to alphabetical column is replaced by concatenation of “C” with the variable’s column number. The code to substitute when using this approach is presented in Figure 19.

```
proc sql;
    CREATE TABLE addressFmt AS
    SELECT strip(upcase(name)) AS start,
           'C'||strip(put(varnum, best.)) AS label length 3,
           "var2Excel" AS fmtName,
           "J" AS type,
           varnum
    FROM sashelp.vcolumn
    WHERE libname = "WORK" AND
          memname = "DATATEMPLATE"
    ORDER BY varNum;
quit;

proc format cntlin = addressFmt;
run;
```

Figure 19. Alternate Step 2 INFORMAT Creation for Export using ODS ExcelXP Tagset

The second change required to export to Excel using the ODS tagset is the creation of the address reference arrays in Step 3, which requires that an “R” is concatenated with the row number. The code to accomplish this is presented in Figure 20. Note that the “rowAdder” macro variable now has a value of one, as the Excel output will begin in the second row.

```

%let rowAdder = 1;

proc sql;
    SELECT strip(put(count(*) + &rowAdder, best.)) INTO :lastRow
    FROM dataTemplate;
quit;

data dataTemplate2;
    length &excel_X &excel_G &excel_R &excel_C $12;
    retain firstBizRow;
    set dataTemplate;
    by business measure;

    array addr_X {*} $ &excel_X;
    array addr_G {*} $ &excel_G;
    array addr_R {*} $ &excel_R;
    array addr_C {*} $ &excel_C;

    firstRow = cats('R', strip(put(1 + &rowAdder, best.)));
    lastRow = cats('R', "&lastRow");
    currentRow = cats('R', strip(put(_N_ + &rowAdder, best.)));
    priorRow = cats('R', strip(put(_N_ + &rowAdder - 1, best.)));
    if first.business then firstBizRow = currentRow;

    do i = 1 to dim(addr_X);

         xlsName = upcase(scan(vname(addr_x(i)), 1, '_'));
         xlsColumn = strip(input(xlsName, $var2excel.));

         addr_x(i) = cats(currentRow, xlsColumn);
         addr_g(i) = cats(firstBizRow, xlsColumn);
         addr_r(i) = cats(addr_g(i), ":", addr_x(i));
         addr_c(i) = compress(cats(firstRow, xlsColumn, ':', lastRow, xlsColumn));


    end;

run;
```

Figure 20. Alternate Step 3 Creation of Excel Address Arrays

CONCLUSION

The replication of SAS analysis in a non-SAS platform, such as Excel, can be a daunting and seemingly meaningless task for a SAS programmer, yet there are occasions when it is advantageous to retain underlying calculations. This paper applies to cases where creating Excel formulas to perform calculations on large amounts of data is tedious or needs to be repeated often, in the hopes of increasing the efficiency and decreasing the frustration of the programmer. By following the simple steps outlined above, demonstrating calculations in Excel will no longer be a burden, but rather a straightforward and efficient way to broaden the audience for a programmer's work.

ACKNOWLEDGMENTS

Thank you to our numerous colleagues who helped to review this paper. You know who you are and your suggestions were very helpful. Any remaining errors are, of course, our responsibility.

RECOMMENDED READING

The export of the data, particularly the details of customizing the appearance of the output, has not been a focus of this paper because there have been numerous papers dedicated to this topic. The following two references provide good examples for formatting Excel data using the ODS excelXP tagsets and DDE, respectively:

- Molter, Michael A. “A Tiptoe Through the Tagset Field.” *Proceedings of the 2008 SAS Global Forum*, Paper 039-2008.
- Watts, Perry. “Using Single-Purpose SAS® Macros to Format EXCEL Spreadsheets with DDE.” *Proceedings of the Thirtieth SAS User Group International Conference*, Paper 089-30.

CONTACT INFORMATION

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

Name: Collin Elliot
Enterprise: Itron, Inc.
Address: 601 Officers Row
City, State ZIP: Vancouver, WA 98661
Work Phone: 360-906-0616
Fax: 360-906-0622
E-mail: collin.elliot@itron.com
Web: www.itron.com

Name: Eli Morris
Enterprise: The Cadmus Group
Address: 720 SW Washington St, Suite 400
City, State ZIP: Portland, OR 97205
Work Phone: 503-228-2992
Fax: 503-228-3696
E-mail: Eli.Morris@cadmusgroup.com
Web: www.cadmusgroup.com