

Paper SBC-137

Parameter-Driven Data Validation and Transformation
Or
The Marriage of Microsoft Excel and SAS® Macro Variables to
Solve ETL Problems
By
Hugh Shinn
Wanda Hopkins

Abstract:

This paper discusses processing data from multiple sources through the use of linked Microsoft Excel parameter files or logical data maps (LDMs) that specify the name, type, length, and validation rules for each target variable in a SAS data set or other table structure (e.g. Oracle, DB2, Access, etc.) and the associated source variable information (e.g. source variable name, and type). The LDM also includes the transformation rules that are used to convert the source variable into the target variable. Each LDM is based on a master logical data map to which there is a link for the validation rules. Thus, it's possible to change the validation rules in the master LDM and have those modifications appear in the subordinate LDMs.

In our case, we examined weather data from North America and Europe to determine their daily variations.

Using SAS Enterprise Guide we imported the LDMs and the input data files. After the imports, we created macro variables from the LDM source variables, transformation rules, target validation rules, and target variables.

Our code determines the number of target variables in the current LDM and, using an iterative %do group, applies the source and transformation rule macro variables to create the target variables. After the target variable creation, the validation rule macro variables are applied.

Using this method, it's possible to change the transformation and validation procedures for specific variables (or add new target variables) without modifying the production code.

Introduction:

The modern world is overrun by data. This is caused in part by increased computer usage, but also by our desire to describe and categorize the world around us.

Data appear in a wide variety of formats and must be normalized so that apples may be compared to apples and not kumquats. One method for normalization is the consistent application of a standard set of data handling rules during the transformation and validation phases of data processing.

The specification of these data handling rules often appears in textual documents that are translated into coding specifications by business analysts, project managers, and programmers. An alternative to this translation is the use of a logical data map (LDM). Our LDMs are written in Microsoft Excel spreadsheets.

A logical data map specifies (among other things):

1. Source variable locations – Name of the file or data set containing the source variables.
2. Source variable names – Name of the variables on the source file.
3. Source variable types – Character, numeric, etc.
4. Transformation rules – Code that converts the source variable into the target variable.
5. Validation rules – Code that checks the value of the target variables after transformation.
6. Target variable names – Name of the variable created by the transformation rules.
7. Target variable locations – Name of the file or data set containing the target variables.

Logical data maps are designed for specific data sources. In this paper, we used temperature data from the United States, Canada, and Europe. We selected data from the Summer and Winter solstices (December, 21 and June, 21) for the years 1993 – 2007.

This combination of U.S. and non-U.S. data sources dictated the use of multiple LDMs because the U.S. temperatures were recorded in degrees Fahrenheit and the non-U.S. temperatures were recorded in degrees Celsius. While it is possible (and sometimes desirable) to combine LDMs, we chose, for the sake of simplicity, to keep them separate.

The transformation and validation rules in our LDMs contain actual SAS statements instead of English explanations. As a result, the rules can be imported into the SAS programs without a parsing process; which can introduce errors. We imported the LDMs into SAS Enterprise Guide 3.0.2 with the import wizard (proc import could be used as well).

We placed a dummy record containing the longest transformation and validation rules in the first row of the LDM. We replaced each letter of the rule with an 'X' to avoid confusing these dummy entries with real rules. This dummy record is read by the SAS import wizard or procedure to determine the length of the transformation and validation rule variables.

Sample of a Logical Data Map:


```

/*****
Program: nbr of ldms
*****/

data ldm;
set sashelp.vstable;
where index(lowercase(memname), 'weather_ldm') > 0;
run;

data _null_;
set ldm end=last;
call symput('ldm' || trim(left(_n_)), trim(left(memname)));
if last then
    call symput('numldms', trim(left(_n_)));
run;

%put ldm1 = &ldm1;
%put ldm2 = &ldm2;
%put numldms = &numldms;

/*****
Program: nbr of data files
*****/

data ds;
set sashelp.vstable;
where index(lowercase(memname), 'us_weather') > 0;
if index(lowercase(memname), 'weather_ldm') > 0 then
    delete;
run;

data _null_;
set ds end=last;
call symput('ds' || trim(left(_n_)), trim(left(memname)));
if last then
    call symput('numds', trim(left(_n_)));
run;

%put ds1 = &ds1;
%put ds2 = &ds2;
%put numds = &numds;

```

Running these code modules will macro-ize the names of the weather data sets and LDMs so we can process them with standard SAS-macro statements or call routines. The macro-based section of the process appears first, followed by the call routine approach. Both approaches yield the same results.

The SAS Macro Approach:

Our first use of SAS macro processing in this project was to delete the dummy records from the LDM data sets. The code below uses the sequential LDM libref and data set macro variables to loop through the data step.

```

/*****
Program: process ldms
*****/

%macro procldm;
  %do i = 1 %to &numldms;
    data &&libref_ldm&i...&&ldm&i;
      set &&libref_ldm&i...&&ldm&i;
      if column_name = 'dummy' then
        delete;
      run;
  %end;
%mend procldm;

%procldm;

```

The next task is to read the transformation and validation rules from the LDM data sets, place them in sequential macro variables, and later into user-defined SAS macro functions.

The transformation rules that are read from the LDMs contain SAS statements and semi-colons. These rules are assigned to sequential SAS macro variables. We did this with a simple data step that reads the LDM data sets and deletes variables that are assigned to the 'weather_summ' data set. The macro variables trans_rules1, trans_rules2, etc. are created using the transformation_rules variable from the LDM, call symput, and the automatic record variable '_n_'. A similar data step is used to create the validation macro variables val_rules1, val_rules2, etc. After creating the sequential macro variables, we used iterative macro do loops to place them in the macro functions trans_mac and val_mac.

The trans_mac macro resolves to a series of SAS statements in the form of X = Y and more complex structures in the form of select-when statements.

Examples:

```

south_longitude = slong;

select (stn);
  when ('BOS') area_name = 'Boston - USA';
  when ('DCA') area_name = 'Washington - DC - USA';
  otherwise;
end;

```

The val_mac macro resolves to a series of if-then-else statements that evaluate an argument, set a variable, and specify a do block.

They take the form of: if A in (B) then X = 'Y' else do; X = 'N'; output dataset; end;

Examples:

```

if area_name in ('SAN','SFO','SEA') then valid='Y' else do; valid= 'N'; output invalid; end;

if 1993 <= year(date) <= 2007 and month(date) in (6,12) and day(date) = 21; else do; valid= 'N';
output invalid; end;

```

Our validation code traps errors and writes them to the data set 'invalid'. This is very useful because it provides a record of data errors.

We completed the data step transformation and validation process with another data step that creates the data set 'validated'. It is created by reading the data sets in the ds macro variables and then calling the trans_mac and val_mac macros.

The data set validated is appended to the data set 'weather_dtl' (referenced in the LDMs). This completes one iteration of the macro read_dtl. These processes are specified in the code below:

```
/******  
Program: process data  
*****/  
  
proc datasets lib=work nolist;  
delete weather_dtl proc_t weather_summ all_invalid;  
quit;  
  
%macro read_dtl;  
  %do i = 1 %to &numds;  
  
/* Identify data step transformation rules from all ldm data sets. */  
  
    data tran;  
    set &&libref_ldm&i...&&ldm&i;  
    if field_no = . or target_table_name = 'weather_summ' then  
        delete;  
  
    column_name = trim(left(column_name));  
  
    if transformation_rules = ' ' then  
        transformation_rules = trim(left(column_name)) || ' = ' || trim(left(source_column_name)) || ';' ;  
    transformation_rules = trim(left(transformation_rules));  
    run;  
  
    options symbolgen;  
  
/* Create data step transformation_rules macro variables. */  
  
    data _null_;  
    set tran end=last;  
  
    call symput('trans_rules' || trim(left(_n_)), trim(left(transformation_rules)));  
    if last then  
        call symput('numtrans',trim(left(_n_)));  
    run;  
  
/* Identify data step validation rules from all ldm data sets. */  
  
    data val;  
    set &&libref_ldm&i...&&ldm&i;  
    if field_no = . or target_table_name = 'weather_summ' or validation_rules = 'none' then  
        delete;  
  
    column_name = trim(left(column_name));  
    validation_rules = trim(left(validation_rules));  
    run;  
  
/* Create data step validation_rules macro variables. */  
  
    data _null_;  
    set val end=last;  
  
    call symput('val_rules' || trim(left(_n_)), trim(left(validation_rules)));  
    if last then  
        call symput('numval',trim(left(_n_)));  
    run;
```

```

/* Create macros trans_mac and val_mac. */

%macro trans_mac;
  %do m = 1 %to &numtrans;
    &&trans_rules&m;
  %end;
%mend trans_mac;

%macro val_mac;
  %do n = 1 %to &numval;
    &&val_rules&n;
  %end;
%mend val_mac;

/* Create data set validated from all ds data sets and call the macros trans_mac and val_mac. */

data validated (drop=tmpf tmpc dwpf dwpc)
  invalid (drop=tmpf tmpc dwpf dwpc);
length area_name $30 region_name $26;
set &&libref_ds&i...&&ds&i;
%trans_mac;
%val_mac;
format date mmddyy10.;
if valid = 'Y' then
  output validated;
run;

/* Create data sets weather_dtl and all_invalid. */

proc append base=weather_dtl data = validated;
run;

proc append base=all_invalid data=invalid;
run;
%end;

%mend read_dtl;

%read_dtl;

/* Create data set proc_t which contains only the procedure step transformation rules */

%macro proc_t;
  %do o = 1 %to &numldms;
    proc append base=proc_t data=&&ldm&o (where=(index(lowercase(transformation_rules),'proc') > 0));
  run;
  %end;
%mend proc_t;

%proc_t;

proc sort data=proc_t nodupkey;
by transformation_rules;
run;

/* Create sequential macro variables proc_trans and macro variable nbr_proc_trans. */

data _null_;
set proc_t end=last;

call symput('proc_trans' || trim(left(_n_)), trim(left(transformation_rules)));
if last then
  call symput('nbr_proc_trans',trim(left(_n_)));
run;

/* Execute sequential macro variables proc_trans and append results to data set weather_summ. */

%macro run_summ;
  %do j = 1 %to &nbr_proc_trans;
    %bquote(&&proc_trans&j);
  %end;

```

```

proc append base=weather_summ (drop=_type_ _freq_) data=summ_out(where=( _type_=31));
run;

%end;
%mend run_summ;

%run_summ;

```

The proc_t transformation rule (see code above) is an entire procedure step (proc means in this case) that processes the data set 'weather_dtl' and produces the data set 'weather_summ' (referenced in the LDMS). In this case, all of the variables in the 'weather_summ' data set (e.g. date, region_name, area_name, south_longitude, south_latitude, temp_lo_f and temp_hi_f) are produced by either the proc means class or var statements.

After completing all of the transformation and validation procedures, we displayed the data with the following gplot program (fig 1 – fig 4). This program also creates the derived variable 'solstice', which is used as a 'by variable' to organize the data.

```

/*****
Program: display data
*****/

data weather_srt;
set weather_summ;
length solstice $6;
if month(date) = 6 then
    solstice = 'Summer';
else
    if month(date) = 12 then
        solstice = 'Winter';
run;

proc sort data=weather_srt;
by region_name solstice;
run;

options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftitle=swissb ftext=swiss htitle=6 htext=3;

symbol1 h=.06in interpol=join v=dot;
symbol2 interpol=join;
symbol3 interpol=join;
symbol4 interpol=join;
symbol5 interpol=join;
symbol6 interpol=join;
symbol7 interpol=join;
symbol8 interpol=join;
symbol9 interpol=join;
symbol10 interpol=join;

axis1 label=none
      order = 1 to 12 by 1
      offset=(2)
      width=3;
axis1 value = (angle= -90) color=black label=none;

proc gplot data=weather_srt;
plot temp_hi_f*date=area_name / haxis=axis1 hminor=0
                               vaxis=axis2 vminor=1
                               caxis=red legend=legend1;
by region_name solstice;
run;
quit;

```

Some of the data results are displayed below:



Enterprise Guide®

The Power to Know.

region_name=Europe - Central solstice=Summer

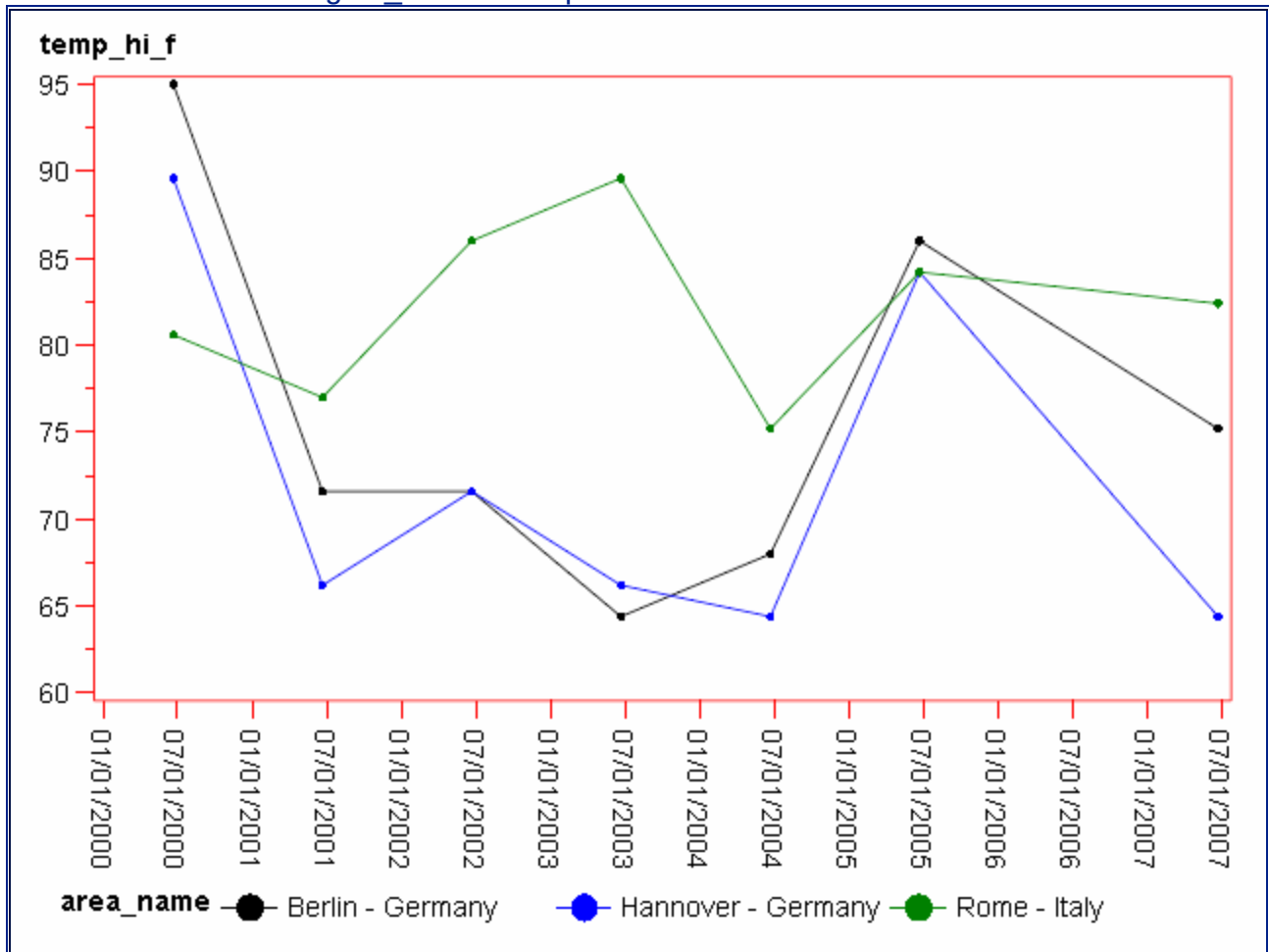


Figure 1

region_name=Europe – Central solstice=Winter

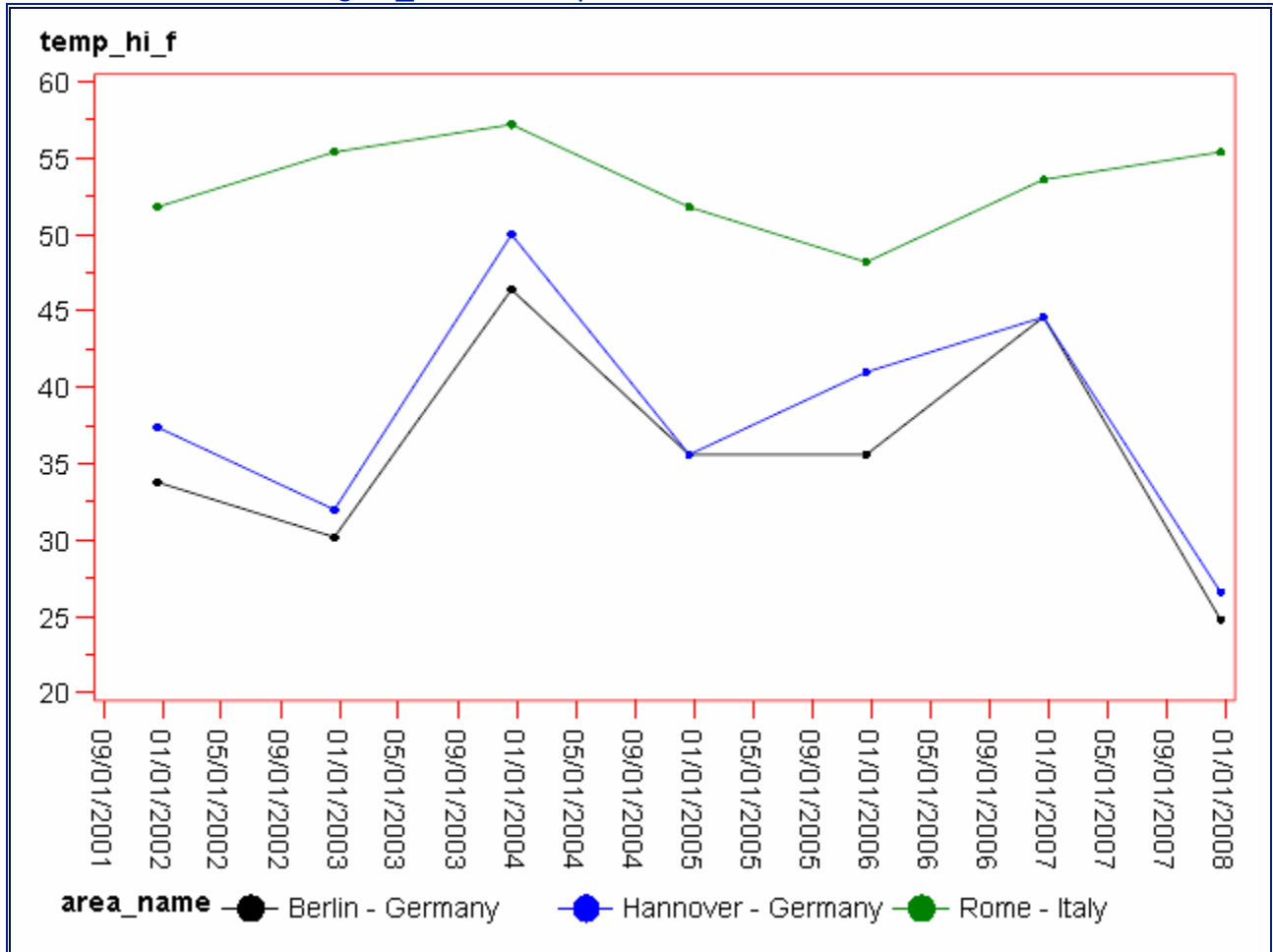


Figure 2

region_name=N. America - Eastern solstice=Winter

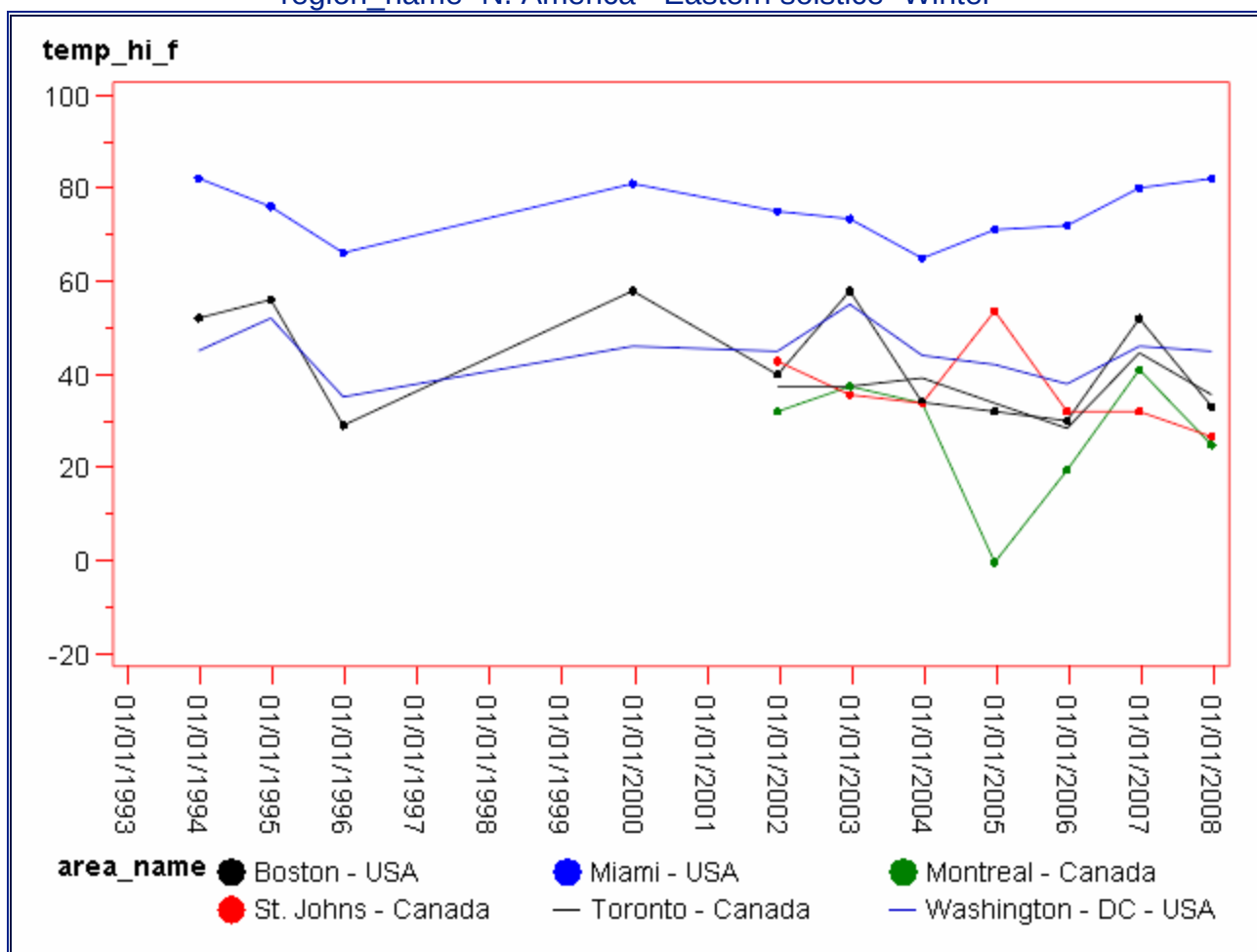


Figure 3

region_name=N. America - Western solstice=Winter

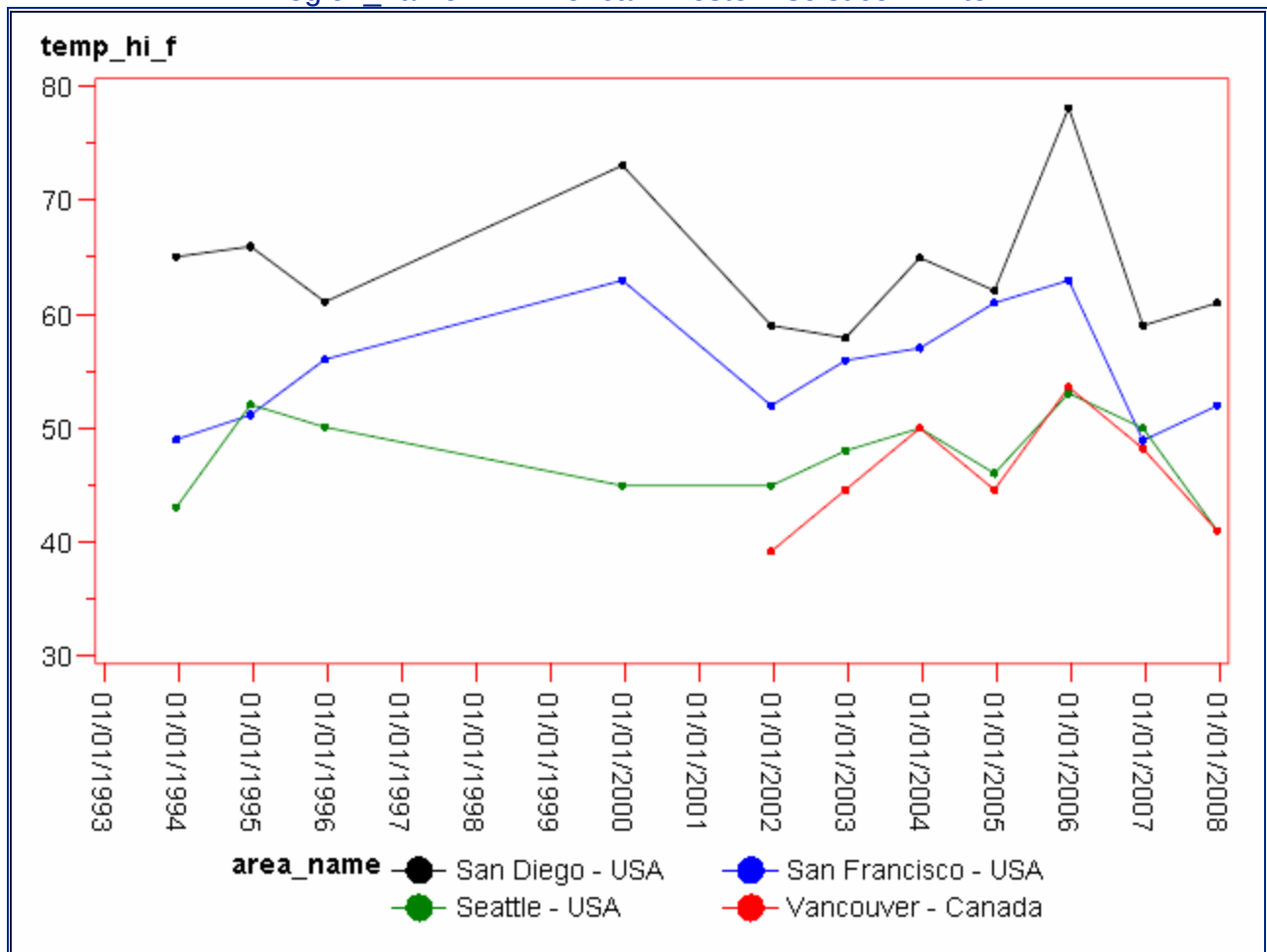


Figure 4

The Call Execute Approach:

Call Execute is a SAS call routine that accepts SAS program statements as parameters. When the current data step ends, the statements passed to Call Execute are compiled and run independently. An alternative method to assigning the SAS logic to macro variables is using Call Execute to build the data step code needed to perform the transformations and validations. Continuing with the same data and logic needed to process the weather data, the first task is to check that all the fields specified in the logical data map are on the source file. This is accomplished with the following code:

```
/* PROGRAM: CHECK_MISSING_VARS.SAS */
```

```

data ldm;
    length dataset_name $ 24;
    set us_weather_ldm(where=(raw_or_calc="RAW"));
run;

proc sort data=ldm;
    by source_column_name;
run;

proc contents data=all_us_data out=contents;
run;

* RETURN ANY FIELDS ON LDM THAT ARE NOT IN SOURCE DATA;
proc sql;
    create table missing_var as
        select a.* from ldm a left join contents b on lowercase(a.source_column_name) = lowercase(b.name)
            where b.name is NULL;
quit;

```

Ideally, this 'missing_var' data set is created with no observations, indicating that there is a match for every field on the logical data map in the source data.

Next, we proceeded to employ the use of Call Execute to transform our raw data using the following code:

```

data _null_;
    set us_weather_ldm (where=(target_table_name="weather_dtl")) end=eof;
    if _n_ = 1 then do;
        call execute('data transformed_data;');
        call execute("set all_us_data;");
        call execute ('length area_name $ 30 region_name $ 26;');
        call execute("format date mmddyy10.;");
    end;
    if raw_or_calc = "RAW" then
        call execute(trim(column_name)||"="||trim(compress(source_column_name, " -")||";"));
        else call execute(trim(transformation_rules));
    if eof then call execute('run;');
run;

```

Within the frame of a data _null_, the logical data map (e.g. 'us_weather_ldm') is read with a where clause to process only records related to the weather_dtl dataset. The call execute statements are used to construct SAS code that will ultimately process the all_us_data data set. The first if statement block generates the statements that only need to be specified once, e.g. , the data statement. Then, for each observation in the 'all_us_data data' set, transformation logic is created by traversing through the list of transformation rules in the 'us_weather_ldm' data set. The eof indicator is used to add statements that will end the building of the data step, e.g. the run statement.

The statements above will produce an execution log similar to the following:

```

11      data _null_;
12          set us_weather_ldm (where=(target_table_name="weather_dtl")) end=eof;
13          if _n_ = 1 then do;
14              call execute('data transformed_data;');
15              call execute("set all_us_data;");
16              call execute ('length area_name $ 30 region_name $ 26;');
17              call execute("format date mmddyy10.;");
18          end;
19
20          if raw_or_calc = "RAW" then
21              call execute(trim(column_name)||"="||trim(compress(source_column_name, " -
22              ")||";"));
23              else call execute(trim(transformation_rules));
24              if eof then call execute('run;');

```

NOTE: There were 7 observations read from the data set WORK.US_WEATHER_LDM.
WHERE target_table_name='weather_dtl';

If the option source2 is turned on you will also see the statements generated by Call Execute prefixed with a plus sign, as shown in the log below:

NOTE: CALL EXECUTE generated line.

```
1      + data transformed_data;  
2      + set all_us_data;  
3      + length area_name $ 30 region_name $ 26;  
4      + format date mmddyy10.;  
5      + date = input(substr(dt_time,1,6),yymmdd10.);  
6      + station_name=STN;  
7      + select (stn);    when ('BOS') area_name = 'Boston - USA';    when ('DCA') area_name =  
'Washington - DC - USA';    when  
( 'DEN') area_name = 'Denver - USA';    when ('GPT') area_name = 'Gulfport-Biloxi - USA';    when ('INL')  
area_name = 'International  
8      + Falls - USA';    when ('MIA') area_name = 'Miami - USA';    when ('MSN') area_name =  
'Madison - WI - USA';    when  
( 'ORD') area_name = 'Chicago - USA';    when ('SAN') area_name = 'San Diego - USA';    when ('SEA')  
area_name = 'Seattle - USA';  
when  
9      + ('SFO') area_name = 'San Francisco - USA';    otherwise; end;  
10     + select;    when (stn in ('BOS','DCA','MIA')) region_name = 'N. America - Eastern';    when (stn  
in ('DEN')) region_name =  
'N. America - Mountain';    when (stn in ('GPT','INL','MSN','ORD')) region_name = 'N. America - Central';  
when (stn in  
11     + ('SAN','SEA','SFO')) region_name = 'N. America - Western';    otherwise ; end;  
12     + south_longitude=SLON;  
13     + south_latitude=SLAT;  
14     + temp_f=TMPF;  
15     + run;
```

NOTE: There were 6124 observations read from the data set WORK.ALL_US_DATA.

NOTE: The data set WORK.TRANSFORMED_DATA has 6124 observations and 19 variables.

Now that all our data has been transformed, we validated it using the validation rules from the logical data map as shown below:

```
data _null_;  
  set us_weather_ldm (where=(target_table_name="weather_dtl" and validation_rules ^= "none"))  
end=eof;  
  if _n_ = 1 then do;  
    call execute('data validated invalid;');  
    call execute("set transformed_data;");  
    call execute("format date mmddyy10.;");  
  end;  
  call execute(validation_rules);  
  if eof then call execute("if valid='Y' then output validated;");  
  if eof then call execute('run;');  
run;
```

Using the same framework, the validation rules are read in from the 'us_weather_ldm' data set and Call Execute statements are used to generate logic that validates fields in the 'transformed_data' data set. In this case, two output data sets are created: 'validated', which contains all records that passed validation and, 'invalid' which contains all error conditions trapped by the validation process.

Note that the fourth Call Execute statement does not contain quoted literal SAS statements as parameters, but instead references the variable 'validation_rules', which when passed to the compiler will translate to the value of the 'validation_rules' variable. It is this ability to pass literal strings as well as variables to Call Execute that allowed us to more fully exploit Call Execute in the design of this project. The code above produces the following log:

NOTE: CALL EXECUTE generated line.

```

1      + data validated invalid;
2      + set transformed_data;
3      + format date mmddyy10.;
4      + if 1993 <= year(date) <= 2007 and month(date) in (6,12) and day(date) = 21 then valid = 'Y';
else do; valid = 'N';
output invalid; end;
5      +
6      +
7      + if station_name in
('BOS','DCA','DEN','GPT','INL','MIA','MSN','ORD','SAN','SEA','SFO','YOW','YTZ','YUL','YVR','YYT','EDDI','E
DDV','EGLL','EIDW','LEMD',
'LFML','LFPO','LIRF') then valid = 'Y'; else do; valid = 'N'; output invalid; end;
8      +
9      +
10     + if area_name in('Boston - USA','Chicago - USA','Denver - USA','Gulfport-Biloxi -
USA','International Falls -
USA','Madison - WI - USA','Miami - USA','San Diego - USA','San Francisco - USA','Seattle -
USA','Washington - DC - USA','Berlin -
Germany','Dublin
11     + - Ireland','Hannover - Germany','London - England','Madrid - Spain','Marseille -
France','Montreal - Canada','Ottawa -
Canada','Paris - France','Rome - Italy','St. Johns - Canada','Toronto - Canada','Vancouver - Canada')
then valid = 'Y'; else do;
valid
12     + = 'N'; output invalid; end;
13     + if region_name in ('N. America - Central','N. America - Eastern','N. America - Mountain','N.
America - Western',
- Central','Europe - Western') then valid = 'Y'; else do; valid = 'N'; output invalid; end;
14     +
15     +
16     + if -20 <= temp_f <= 100 then valid = 'Y'; else do; valid = 'N'; output invalid; end;
17     +
18     +
19     + if valid='Y' then output validated;
20     + run;

```

NOTE: There were 6124 observations read from the data set WORK.TRANSFORMED_DATA.

NOTE: The data set WORK.VALIDATED has 5928 observations and 20 variables.

NOTE: The data set WORK.INVALID has 200 observations and 20 variables.

Conclusions:

Whether using macro variables or Call Execute, the use of logical data maps provides an autonomous, low-maintenance method of developing standardized, and uniform code that can be used over a variety of applications that are dependent on diverse source data. Having the transformation and validation rules standardized in a single document significantly reduces the time needed for individual programmers to design, code, and test from scratch and insures that the identical processes are being run against mission-critical data. The logical data map also serves as a sort of data repository and the developer has at her disposal all relevant metadata for the task at hand.

Another advantage to using logical data maps is that code changes for production systems are relatively simple. It's no longer necessary to check the programs out of production and search for the appropriate section of code to change.

However, if you're working in an industry that has lots of oversight (e.g. Financial Services) it will be necessary to add the columns 'change-date', 'changed-by', 'change-reason' and 'change' to the LDM so that auditors will be able to track system modifications. You'll also have to store the logical data map in a secure location.

Notes:

This is not a climate study. The weather data presented in this paper are included for illustrative purposes only.

References:

The weather data used in this paper was provided by the National Severe Storms Laboratory Historical Weather Data Archives, Norman, Oklahoma, from their Web site at <http://data.nssl.noaa.gov>".

The weather data prior to April 20, 1998 were provided to NSSL by:

Robbins, C. C., and J. V. Cortinas Jr., 2002: Local and synoptic environments associated with freezing rain in the contiguous United States. *Wea. Forecasting* , 17, 47 - 65.

Cortinas, J. V. Jr., B. C. Bernstein, C. C. Robbins, and J. W. Strapp, 2004: An analysis of freezing rain, freezing drizzle, and ice pellets across the United States and Canada: 1976 - 90. *Wea. Forecasting* , 19, 377 - 390.

Acknowledgements:

Thank you to David Martinez for his assistance with the Gplot procedure.

Contact Information:

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

Hugh Shinn
Phone: 730-343-0405 (Cell)
Email: hughshinn@yahoo.com

Wanda Hopkins
Phone: 703-236-8668
Email: Wanda.Hopkins@etrade.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.