Paper SAS153-2014

SAS® Format Optimization: SAS_PUT or UNPUT (Who's On First?)

David Wiehle, SAS Institute Inc., Cary, NC

ABSTRACT

Changes in default behavior in the last few SAS releases have allowed for faster processing of SAS formats, especially for SAS/ACCESS® customers. But, as with any performance enhancement, your results might vary.

This paper describes:

- the differences between two important SAS format optimizations.
- how to tell which optimization is in effect.
- a simple method to get the behavior you want.

The target audience for this paper includes SAS/ACCESS customers, particularly those who have also licensed the SAS® Scoring Accelerator for Teradata, DB2, Netezza, Greenplum, or Aster.

INTRODUCTION

The origin of this paper was my own frustration when presented with this technology for the first time. The similarity in the names of the optimizations and the options that control them was almost comical ("Where should I put this UNPUT? Over there by the SAS_PUT? Do I need to reduce PUT or map PUT?"). It became clear this was a case where terminology simply got in the way. This paper turns that problem on its head by temporarily setting aside the terminology and focusing on the behavior first.

Why should you use format optimization? Consider the following scenario. I created an input table in Teradata with 8 million observations. I wrote a simple SQL query with a PUT in the SELECT and the WHERE, which resulted in a 640,000 observation result set. Using the techniques explained in this paper, I was able to reduce what initially took 21 seconds without optimization to 2.4 seconds with optimization.

FORMAT OPTIMIZATION BASICS

Format optimization was designed to improve performance of a SAS process. SAS/ACCESS customers are familiar with the performance cost of pulling a large DBMS table into SAS to execute a query. In the initial releases of SAS/ACCESS, queries or procedures that referenced a SAS format would cause all the input observations to be pulled into the SAS session for processing. Then the results would be pushed back down to the DBMS. Format optimization was developed in conjunction with SQL optimizations such as implicit pass-through to reduce data movement and improve performance.

Both user-defined and intrinsic (or supplied by SAS) formats are supported. For example:

```
put(sample_value, $myformat.);
put(sample_date, date9.);
format sample_value $myformat.;
```

The first two lines above might be found in a PROC SQL query. The third line represents a FORMAT statement that might appear in a Base SAS procedure. Certain SAS procedures support in-database processing, in which the procedure is converted to DBMS-specific SQL, and passed down for processing by the DBMS.

For the purposes of this introduction, the optimizations are referenced by their relationship to the SAS session: "Local" refers to the optimization of formats in the SAS session. Format values are translated by the SQL textualizer into SQL "CASE/WHEN/ELSE" statements. "Published" optimization involves the definition of a SAS format as a user-defined DBMS function in a specially staged server. Local optimization is possible for all customers, (even for BASE engine input tables), but published optimization is possible only for certain DBMSs after installing software provided by SAS.

Format publishing will be more fully explained in Appendix 1. At first glance, it might appear that published optimization is out of reach for most customers since it requires installing software onto a DBMS server and one-time

publishing of formats to the DBMS. If you are already using the SAS-Scoring Accelerator, you most likely already have access to this technology. However, due to the manner in which this technology is bundled with the SAS/ACCESS products, *all* SAS/ACCESS customers for the DBMSs listed in the abstract have already bought and paid for this technology. If you are interested in maximizing the return on your investment, it might be well worth your time to discuss this technology with your DBA.

The examples below are valid for a SAS 9.3 or later in-database environment. To simplify the examples, it is presumed that the DBMS has been staged with the required installed components to support published optimization and the formats have already been published to the DBMS. In addition, it is presumed that the SQL query or the in-database procedure uses syntax that allows processing by the DBMS.

3-3-1 METHOD

The 3-3-1 Method is a simple way to learn (and remember) how to make each optimization work.

- three behaviors controlled by
- three system options and
- one format attribute

This method does not guarantee a particular result, but rather puts you in command of the SAS session so that SAS attempts a particular optimization.

3-3-1 METHOD: THREE BEHAVIORS

The presence of a PUT statement in an SQL query or a FORMAT statement in an in-database procedure can result in one of three behaviors:

- no optimization
- local optimization
- published optimization

No optimization SAS performs all the query processing

Local optimization: SAS attempts to convert the format values to “CASE/WHEN/ELSE” statements and pushes them down to the DBMS for execution.

Published optimization: SAS attempts to use the version of the format published to the DBMS as a user-defined function.

3-3-1 METHOD: THREE SYSTEM OPTIONS

The three behaviors are above are controlled by three system options. Two of the options below can also interact with SQL procedure options, and the system options have more values than are described below. For the purposes of simplicity, only the system options are listed, and only those option values that are necessary to trigger the optimizations are described below. The default option values are in bold.

- SQLREDUCEPUT = none | **DBMS**
- SQLMAPPUTTO = none | **SAS_PUT**
- SQLGENERATION = none | **DBMS**

SQLREDUCEPUT: Jargon-free translation: “Should SAS attempt to convert SAS formats to CASE/WHEN/ELSE statements?”

- none = “No. Use the formats as-is in the SAS session.”
- DBMS = “Yes. Convert the formats to CASE/WHEN/ELSE statements, push them down to the DBMS for processing”

SQLMAPPUTTO: Jargon-free translation: “Where should SAS search for formats?”

- None = “In the SAS session.”
- SAS_PUT = “In the DBMS”

  Note that for DB2, the option value is SQLMAPPUTTO=(*schema*.SAS_PUT) , where *schema* is the name of the schema in which the formats were published for this particular database.

SQLGENERATION: Jargon-free translation: " Should SAS generate DBMS-specific SQL for certain SAS procedures?"

- none = "No"
- DBMS = "Yes"

There are six DBMSs that support in-database processing of procedures in SAS 9.3: Aster, DB2, Greenplum, Netezza, Oracle, and Teradata. Hadoop was added in SAS 9.4. There are seven SAS procedures that support this feature: FREQ, MEANS, RANK, REPORT, SORT, SUMMARY, and TABULATE.

3-3-1 METHOD: ONE FORMAT ATTRIBUTE

The final element of the 3-3-1 method requires a little background. Local format optimization translates the values of a SAS format into DBMS-specific SQL. The SAS format is converted into "CASE/WHEN/ELSE" statements that are "understood" by the DBMS. For example:

```
case
when "myvar" = '0' then 'zero'
when "myvar" = '1' then 'one'
when "myvar" = '2' then 'two'
else 'Other'
```

In order for SAS to attempt this optimization, the full range of possible outcomes for a format must be available. Consider this user-defined format and a simple SELECT query that uses the format:

```
proc format;
value $isotype
  'Ag-110m'='Silver'
  'Am-241'='Americium'
  'Ba-140'='Barium';
run;

proc sql;
select put(myvar, $isotope.) as newvar
from DBLIB.mytable;
quit;
```

In this example, SAS has only part of the information necessary to construct the DBMS-specific SQL necessary for this format. SAS can build the "WHEN" portion:

```
when "myvar" = 'Ag-110m' then 'Silver'
when "myvar" = 'Am-241' then 'Americium'
when "myvar" = 'Ba-140' then 'Barium'
```

But SAS does not have a "default" value to use in an ELSE clause. SAS can do this if the format is redefined using an OTHER= clause. OTHER= is essentially the default label for non-matching data values.

```
proc format;
value $isotypeo
  'Ag-110m'='Silver'
  'Am-241'='Americium'
  'Ba-140'='Barium'
  other='Not specified';
run;

proc sql;
select put(myvar, $isotopeo.) as newvar
from DBLIB.mytable;
quit;


when "myvar" = 'Ag-110m' then 'Silver'
when "myvar" = 'Am-241' then 'Americium'
```

```
when "myvar" = 'Ba-140' then 'Barium'
else 'Not specified'
```

Unfortunately, this is not an absolute rule. Due to the inner workings of the SQL textualizer, it is possible for SAS to attempt local optimization for formats without an OTHER= value if the PUT appears in a WHERE or HAVING clause. However, as a best practice, it is recommended that an OTHER= value be used in user-defined formats to ensure the best performance of local optimization in all situations.

DEFINITIONS OF FORMAT OPTIMIZATIONS

(This is the “I cannot stall any longer” section).

- Local optimization is **UNPUT**: SAS attempts to convert the format values to CASE/WHEN/ELSE statements and pushes them down the DBMS for execution.
- Published optimization is **SAS_PUT**: SAS attempts to use the version of the format published to the DBMS as a user-defined function.

3-3-1 METHOD: FLOWCHARTS

The flowcharts below demonstrate how the system options and format attributes interact in SAS 9.3 and later.

The three behaviors appear at the bottom of each flowchart.

- No optimization is represented by “NO” in an octagonal box.
- Local optimization is represented by “UNPUT” in an oval.
- Published optimization is represented by “SAS_PUT” in a cylinder.

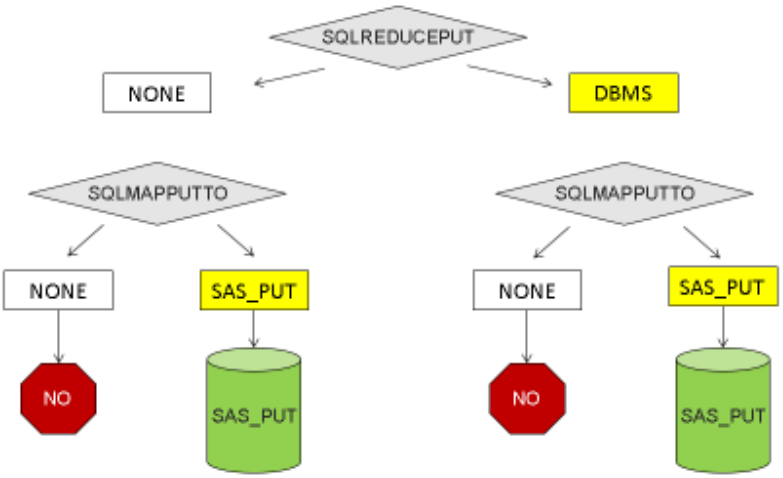
The three system options are in diamond-shaped boxes. The values of the option appear immediately below the option name. The default value of the option is highlighted in yellow. If this document is printed in black and white, the option to the right is the default option. For example, in Figure 1 below, SQLREDUCEPUT appears at the top of the flowchart. Its two values are NONE on the left, and DBMS on the right. DBMS is the default option.

The flowcharts are presented with “limited commercial interruption” for the purposes of clarity. The SAS code and log tracing examples that correspond to each chart appear in the following section.

Figures 1 and 2 below describe the possible optimizations for an SQL query that includes a PUT statement. For example:

```
proc sql;
create table nuclides as
select s_id, nuclide, put(nuclide, $isotype.) as myvar
from dblib.quake8mil
order by s_id;
quit;
```

Figure 1. SQL SELECT Query, Format without an OTHER= Value



The SQLGENERATION= option is not included in Figures 1 and 2 because this option's value has no effect on the optimization of an SQL query. In Figure 1, note that there is no way for SAS to attempt local format optimization (UNPUT). If formats are published to the DBMS, SAS attempts published optimization (SAS_PUT) by default.

Figure 2. SQL SELECT Query, Format with an OTHER= Value

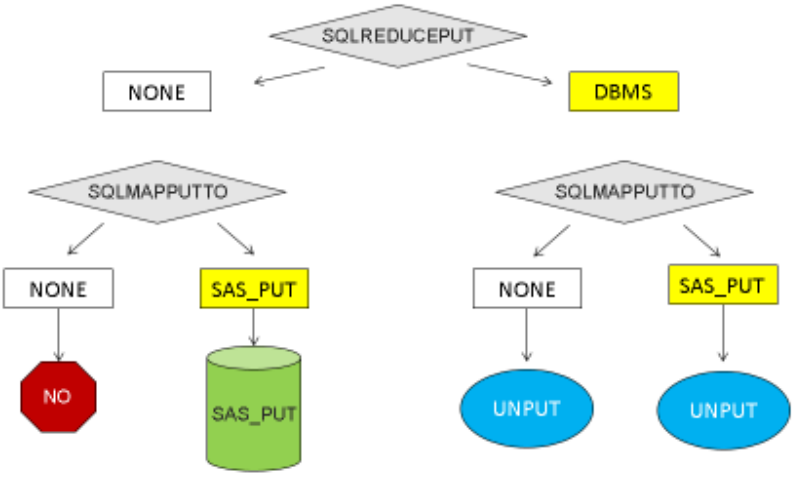
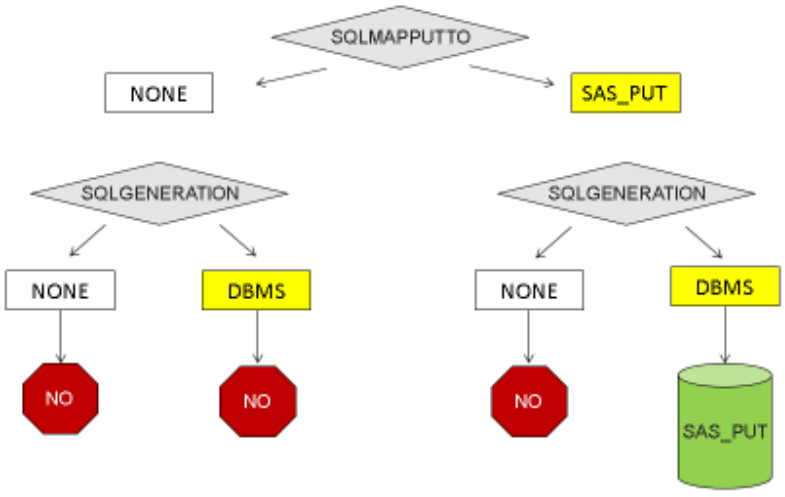


Figure 2 illustrates the value of the 3-3-1 method. In this scenario, if default values are used for SQLREDUCEPUT= and SQLMAPPUTTO=, that is, SAS is instructed to optimize both the local and published formats, SAS has to choose which option to attempt. In this situation, SAS attempts local optimization (UNPUT). If you wish to use published optimization (SAS_PUT), the only way to ensure that SAS attempts this optimization is to set SQLREDUCEPUT=NONE, the non-default value.

Figures 3, 4 and 5 describe the possible optimizations for an In-database procedure that contains a format. Here is an example:

```
proc freq data=dblib.dfmtverf;
format age agfmt.;
tables age age*month / list bin;
run;
```

Figure 3. In-database Procedure, Format with an OTHER= Value SQLREDUCEPUT=NONE



The scenario in Figure 3 begins with the non-default setting of SQLREDUCEPUT=. Note that the only possible optimization in this scenario is published optimization (SAS_PUT).

Figure 4. In-database Procedure
Format with an OTHER= value SQLREDUCEPUT=DBMS

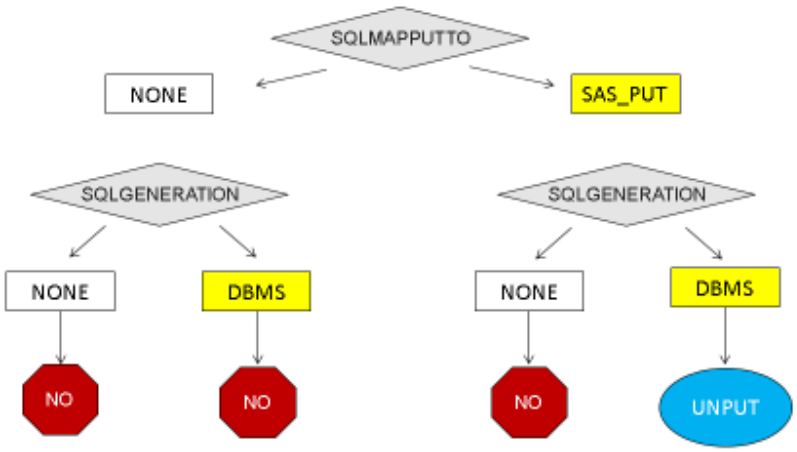
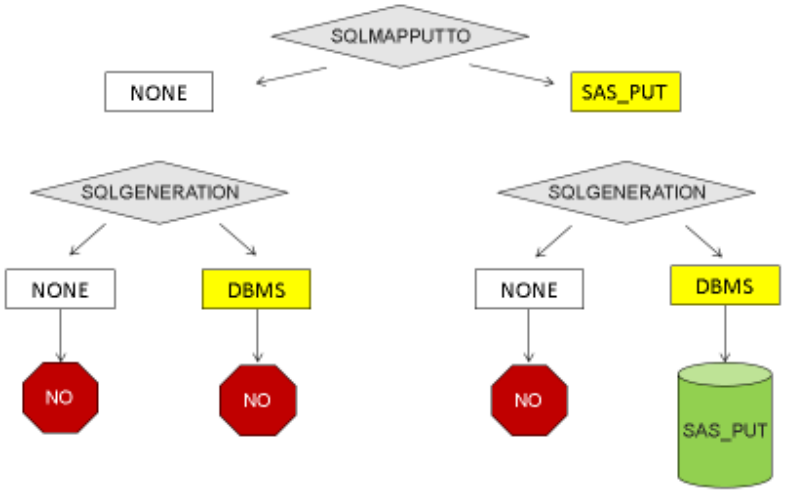


Figure 4 is the mirror image of Figure 3. The scenario begins with the default setting of SQLREDUCEPUT=, and this results in the exact opposite behavior: The only possible optimization is local optimization (UNPUT).

Figure 5. In-database Procedure
Format without an OTHER= value



The behavior described in Figure 5 is identical to Figure 3 because the format does not have an OTHER= value. As in Figure 3, the only possible optimization is published optimization (SAS_PUT).

SAS LOG TRACING: HOW-TO

(The "Trust but verify" section). The flowcharts above illustrate how SAS determines what optimization to attempt. Two levels of tracing are suggested to prove what happened after SAS made that attempt:

- **Debugging**: Four system options provide a great deal of information about the nature of the SQL generated on your behalf.
- **Production:** Two system macro variables output log notes indicating when local optimization (UNPUT) and/or published optimization (SAS_PUT) is in effect

DEBUGGING

As indicated in the introduction, format optimization was developed in conjunction with SQL implicit pass-through. The debugging-level log tracing options below are very similar to the options used for that technique.

```
options sastrace=',,,d'
      sastraceloc=saslog
      sql_ip_trace=(note,source)
      msglevel=i;
```

These options are in fairly heavy use for SAS/ACCESS customers, and are not be fully described here. However, in the context of format optimization, the options

SASTRACE and SASTRACELOC display all SQL statements that are sent to the DBMS. For example:

```
TERADATA: Executed: on connection
 SQL  select TXT_1."s_id", TXT_1."collection_date", TXT_1."nuclide",
```

SQL_IP-TRACE displays information about the SQL generated by SAS. For example:

```
SQL_IP_TRACE: pushdown attempt # 1
SQL_IP_TRACE: The SELECT statement was passed to the DBMS
```

MSGLEVEL=i displays messages about published formats . For example:

```
NOTE: XOG: Put Ping Query
```

It is recommended that, whenever possible, tracing options be "turned on" prior to a query or procedure, then "turned off" after processing. For example:

```
options sastrace=',,,d'
      sastraceloc=saslog
      sql_ip_trace=(note,source)
      msglevel=i;

proc sql;quit;

options sastrace=',,,,'
      sql_ip_trace=none
      msglevel=n;
```

PRODUCTION

Like most DBMS tracing, the log output produced by the debugging options above can be cumbersome for production jobs. I prefer a straightforward "Yes, it worked" message. I requested the options below be created to streamline the review of my query and procedure results. These options are recommended for use to simply the review of your production jobs.

%let SYS_SQL_UNPUT_TRACE=1;

```
PROC SQL performs UNPUT processing to avoid format $ISOTYPEO11.
```

%let SYS_SQL_PUT_TRACE=1;

```
NOTE: PROC SQL found the format MYRANGE.and SAS_PUT will be generated for it.
```

If you wish to "turn off" these options, simply set the macro variable to a value other than 1. For example:

```
%let SYS_SQL_PUT_TRACE=0;
```

SAS LOG TRACING: FORMAT OPTIMIZATION IN ACTION

So what do each of the optimizations look like?

The examples below are drawn from the demo program in Appendix 2b. They use a PROC SQL query of a Teradata input table, which uses a PUT statement in a SELECT and a WHERE:

```
proc sql ;
create table WORK.results as
select s_id, collection_date,
     nuclide,  put(nuclide, $isotypeo.) as myvar,
     s_result, put(s_result, rangec.) as myvar2
from dblib.quake8mil
where put(nuclide, $isotypeo.)='Strontium' and
         index(left(trim(put(s_result, rangec.))), 'Alert') > 0
order by s_id
;
quit;
```

The user-defined formats in the PUT statements were both defined with an OTHER=value. For example:

```
proc format;
  value $isotypeo
  'Ag-110m'='Silver'
  'Am-241'='Americium'
  'Ba-140'='Barium'
  'Cs-134'='Caesium'
  other='Other'
  ;
run;
```

For the purposes of readability, the tracing examples are presented without the PROC SQL code that generated the tracing. Each example begins with the system options used to produce the output. The complete demo program is presented in Appendix 2b.

SAS LOG TRACING: LOCAL OPTIMIZATION (UNPUT) CORRESPONDS TO FIGURE 2

```
options sqlgeneration=DBMS
      sqlreduceput=DBMS
      sqlmapputto=none;

/* production log tracing options */
%let SYS_SQL_UNPUT_TRACE=1;
```

```
%let SYS_SQL_PUT_TRACE=1;

PROC SQL performs UNPUT processing to avoid format $ISOTYPEO11.
PROC SQL performs UNPUT processing to avoid format RANGEC25.
PROC SQL performs UNPUT processing to avoid format $ISOTYPEO11.
PROC SQL performs UNPUT processing to avoid format RANGEC25.
```

In this example, SYS_SQL_UNPUT_TRACE outputs a line of output for each occurrence of a format in the query. Since each format appears both in the SELECT and the WHERE, two messages are output for each format.

```
/* debugging tracing options */

options sql_ip_trace=(note, source)
      sastrace=',,,d' sastraceloc=saslog
      msglevel=i;

SQL_IP_TRACE: pushdown attempt # 1
SQL_IP_TRACE: passed down query:   select TXT_1."s_id",
TXT_1."collection_date", TXT_1."nuclide",
case  when TXT_1."nuclide" = 'Ag-110m' then 'Silver' when TXT_1."nuclide" =
'Am-241' then 'Americium'
when TXT_1."nuclide" = 'Ba-140' then 'Barium' when TXT_1."nuclide" = 'Ce-144'
then 'Cerium' when
TXT_1."nuclide" = 'Cs-134' then 'Caesium' when TXT_1."nuclide" = 'Cs-136' then
'Caesium' when
TXT_1."nuclide" = 'Cs-137' then 'Caesium' when TXT_1."nuclide" = 'Gross Alpha'
then 'Gross Alpha'
when TXT_1."nuclide" = 'Gross Beta' then 'Gross Beta' when TXT_1."nuclide" =
'I-131' then 'Iodine'
when TXT_1."nuclide" = 'I-132' then 'Iodine' when TXT_1."nuclide" = 'Mo-99'
then 'Molybdenum' when
TXT_1."nuclide"

SQL_IP_TRACE: The SELECT statement was passed to the DBMS.
```

To simplify this example, the above output was limited to a portion of what is produced by sql_ip_trace=(note, source). Note the CASE/WHEN statements, which correspond to the values of the $isotypeo.” format. This is the dead giveaway that local optimization (UNPUT) is at work. The other key information produced by this option is the fact that the SELECT statement was passed to the DBMS, as well as the number of attempts SAS made to do this, In this case, a single attempt.

SAS LOG TRACING: PUBLISHED OPTIMIZATION (SAS_PUT) CORRESPONDS TO FIGURE 3

```
options sqlgeneration=DBMS
      sqlreduceput=none
      sqlmapputto=DBMS;

/* production log tracing options */
%let SYS_SQL_UNPUT_TRACE=1;
%let SYS_SQL_PUT_TRACE=1;

NOTE: PROC SQL found the format RANGEC25.0 and SAS_PUT will be generated for
it.
NOTE: PROC SQL found the format $ISOTYPEO11.0 and SAS_PUT will be generated for
it.
```

In this example, SYS_SQL_PUT_TRACE outputs at least one line of output for each format.

```
/* debugging tracing options */
```

```
options sql_ip_trace=(note, source)
      sastrace=',,,d' sastraceloc=saslog
      msglevel=i;

NOTE: XOG: Put Ping Query
NOTE: SELECT SAS_PUT('$ISOTYPEO', '$IS-INTRINSIC') AS X, SAS_PUT('$ISOTYPEO',
'$FMT-META') AS Y FROM
     (SELECT COUNT(*) AS C FROM  "model"."quake8mil" WHERE 0=1) A
NOTE: XOG: Put Ping Query
NOTE: SELECT SAS_PUT('RANGEC', '$IS-INTRINSIC') AS X, SAS_PUT('RANGEC', '$FMT-
META') AS Y FROM
     (SELECT COUNT(*) AS C FROM  "model"."quake8mil" WHERE 0=1) A
SQL_IP_TRACE: pushdown attempt # 1
SQL_IP_TRACE: passed down query:   select TXT_1."s_id",
TXT_1."collection_date", TXT_1."nuclide",
cast(SAS_PUT(TXT_1."nuclide", '$ISOTYPEO11.0') as char(11)) as "myvar",
TXT_1."s_result",
cast(SAS_PUT(TXT_1."s_result", 'RANGEC25.0') as char(25)) as "myvar2" from
"model"."quake8mil" TXT_1
where (cast(SAS_PUT(TXT_1."nuclide", '$ISOTYPEO11.0') as char(11)) =
'Strontium') and
(POSITION('Alert' IN TRIM(LEADING FROM TRIM(TRAILING FROM
cast(SAS_PUT(TXT_1."s_result",
'RANGEC25.0') as char(25))) ) )  > 0) order by TXT_1."s_id" asc
SQL_IP_TRACE: The SELECT statement was passed to the DBMS.
```

This excerpt begins with an “XOG Ping” query generated by MSGLEVEL=i. This is an indication that SAS searched for formats in the DBMS, located them, and determined they were user-defined formats. The “sql_ip_trace” output includes “cast(SAS_PUT”, which is an indication that SAS is using the format that was published to the DBMS. The excerpt ends with a note that the SELECT statement was passed to the DBMS.

SAS LOG TRACING: TROUBLESHOOTING TIPS

The examples below demonstrate how SAS log tracing can provide troubleshooting assistance.

```
NOTE: Optimization for the PUT function was skipped because the referenced
format, X3X, does not have an OTHER= range defined.
```

When the SAS system options are tuned to attempt local optimization (UNPUT), SAS produces the above note when it encounters a format that does not have an OTHER= value. To resolve this problem, redefine the format in the SAS session with an OTHER= value and rerun the query.

```
ERROR: The format $ISOTYPEO was not found or could not be loaded.
```

A basic design feature of published optimization (SAS_PUT) is that there be a fallback position if there is some problem processing the format in the DBMS. For this reason, SAS_PUT requires that the format be defined in the local SAS session. SAS produces the above error when system options are tuned to attempt SAS_PUT and the format is not found. To resolve this problem, define the format in the SAS session and rerun the query.

```
NOTE: XOG: Put Ping Query
NOTE: SELECT SAS_PUT('RANGEX', '$IS-INTRINSIC') AS X, SAS_PUT('RANGEX', '$FMT-
META') AS Y FROM
     (SELECT COUNT(*) AS C FROM  "model"."quake8mil" WHERE 0=1) A
NOTE: PROC SQL cannot use the format RANGEX25.0 on the database.
```

The SAS log notes are produced in the exact opposite situation as in the prior example. When SAS system options are tuned to attempt published optimization (SAS_PUT), and the format is not defined in the DBMS but is found in the

local SAS session, SAS produces the above note and processes the format in SAS. An important clue is that the "cannot use the format" note appears immediately after the "XOG ping" note for the same format. Debug level tracing options are required to produce this output.

```
NOTE: XOG: Put Ping Query
NOTE: SELECT SAS_PUT('X3XO', '$IS-INTRINSIC') AS X, SAS_PUT('X3XO', '$FMT-
META') AS Y FROM (SELECT COUNT(*) AS C FROM  "sasrank"."fmt0035" WHERE 0=1) A
ERROR: Format check sum error.
NOTE: PROC SQL cannot use the format X3XO5.0 on the database.
ACCESS ENGINE:  SQL statement was not passed to the DBMS, SAS will do the
processing.
```

What the BLEEP is a "check sum"? When published optimization (SAS_PUT) is attempted, SAS performs a verification in addition to the "is the format present locally?" test described in an earlier example. SAS compares the format values and labels of the local and published versions of the format. If there are any differences, SAS produces the "check sum" error displayed in the example above, and processes the query in SAS using the local format. To resolve this error, redefine the format in the SAS session using the same values and labels as are present in the published version (or vice versa).

BEST PRACTICE SUGGESTIONS

Do not settle for default behavior. Take command of the SAS session and set the system options to attempt the format optimization you want.

SYSTEM OPTIONS FOR LOCAL OPTIMIZATION (UNPUT)

```
options sqlgeneration=DBMS
      sqlreduceput=DBMS
      sqlmapputto=none;
```

SYSTEM OPTIONS FOR PUBLISHED OPTIMIZATION (SAS_PUT)

```
options sqlgeneration=DBMS
      sqlreduceput=none
      sqlmapputto=SAS_PUT;
```

SYSTEM OPTIONS FOR DEBUGGING

```
options sastrace=',,,d'
      sastraceloc=saslog
      sql_ip_trace=(note,source)
      msglevel=i;

proc sql;quit;

options sastrace=',,,,'
      sql_ip_trace=none
      msglevel=n;
```

SYSTEM OPTIONS FOR PRODUCTION JOBS

```
%let SYS_SQL_UNPUT_TRACE=1;
%let SYS_SQL_PUT_TRACE=1;
```

REFERENCES

Whitcher, Mike. 2008. “New SAS® Performance Optimizations to Enhance Your SAS® Client and Solution Access to the Database.” *Proceedings of SAS Global Forum 2008*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/sgf2008/optimization.pdf

SAS Institute Inc. 2014. SAS/ACCESS for Relational Databases: Reference. “Macro Variables and System Options for Relational Databases”. Available at > General Reference > Macro Variables and System Options for Relational Databases http://support.sas.com/documentation/cdl/en/acreldb/65247/HTML/default/viewer.htm#n0yfsgnsierxrsn1197jnrdvv9rp.htm

SAS Institute Inc. 2013. *SAS In-Database Products: User’s Guide*. “Format Publishing and the SAS_PUT ( ) Function” Available at http://support.sas.com/documentation/onlinedoc/indbtech/index.html#indbtech94

US Department of Energy. 2013. US DOE/NNSA and DoD Response to 2011 Fukushima Incident: Radiological Air Samples.Available at http://catalog.data.gov/dataset/us-doennsa-and-dod-response-to-2011-fukushima-incident-radiological-air-samples

APPENDIX 1: HIGH-LEVEL SUMMARY OF FORMAT PUBLISHING

The system requirements and format publishing process varies slightly from DBMS to DBMS. For specifics, please refer to the SAS 9.3 or 9.4 *SAS In-Database Products: User’s Guide* for your DBMS. Instructions can be found under “Deploying and Using SAS formats in [DBMS name]”.

In SAS 9.3 and 9.4, format publishing is supported in the following DBMSs:

- Aster
- DB2 under Unix
- Greenplum
- Netezza
- Teradata

When any of the SAS/ACCESS products listed above are installed, a package of files is downloaded to the SAShome directory. This package of files contains the components necessary to support format publishing. For specifics on these files, please refer to “Deployed Components for In-Database Processing” in the *SAS In-Database Products: User’s Guide*.

High-level summary:

1. DBMS and operating system commands
    a. The package of files is copied to the DBMS server.
    b. DBMS-specific commands are used to deploy the files onto the server.
2. SAS Session 1 (publish formats)
    a. Any desired user-defined SAS formats are defined in the SAS session. If only intrinsic (~~SAS-supplied~~ supplied by SAS) formats are desired, skip to Step 2b.
    b. A SAS macro is run to define the precompiled publishing macro
    c. The publishing macro is run, defining SAS formats as a user-defined function in the DBMS. Once published to the DBMS, the formats can be accessed by any future SAS session without the need to republish.
3. SAS Session 2 (use published formats)
    a. Intrinsic formats can be accessed once the publishing macro is run.
    b. User-defined formats must be defined in the current SAS session using the same format values and format labels as appear in the published version.

APPENDIX 2A: DEMO PROGRAM, INPUT TABLE CREATION

The PROC SQL CONNECT TO code below was written for Teradata, but the data set WORK.test produced by the DATA step below can be directed to any SAS/ACCESS library. The input data is based on field team radiological measurements taken by the US DOE/NNSA, in response to the 2011 Fukushima Incident. A link to the original source data can be found in the "References" section of this paper.

```
/* revise the statements below to match the requirements of your server */
%let dbserver=[server name];
%let dbuser=[user name];
%let dbpw=&dbuser;
%let database=&dbuser;

/* explicit connection used by Proc SQL "connect to" query below */
%let exconn=%str( server=&dbserver user=&dbuser password=&dbuser database=&dbuser );

/* libref used by Proc Append code below */
libname dblib teradata server=&dbserver user=&dbuser pw=&dbpw database=&database;



PROC SQL;
  CONNECT TO TERADATA (&exconn);
  EXECUTE (DROP TABLE quake8mil) by TERADATA;
  EXECUTE (COMMIT) BY TERADATA;
  EXECUTE (CREATE MULTISET TABLE quake8mil ( s_id INTEGER ,
                                 collection_date TIMESTAMP ,
                                 nuclide CHAR(12) ,
                                 s_result DECIMAL(8,6)
                                ) PRIMARY INDEX (s_id)) by TERADATA;
  EXECUTE (COMMIT) BY TERADATA;
QUIT;


/* data creation code */
data test(drop=i j k);
length s_id 8. nuclide $12. ;
format COLLECTION_DATE datetime26.;
retain s_id 0;
do i=1 to 20;

/* create the collection_date */

do j=1 to 200;
collection_date=1636345640+(i*100);


do k=1 to 2000;
s_result=ranuni(14);

/* 26 nuclides */

if 0 lt s_result le .04 then do;
  nuclide='Ag-110m';
end;

if .04 lt s_result le .08 then do;
  nuclide='Am-241';
end;

if .08 lt s_result le .12 then do;
  nuclide='Ba-140';
end;
```

```
if .12 lt s_result le .16 then do;
  nuclide='Ce-144';
end;

if .16 lt s_result le .20 then do;
  nuclide='Cs-134';
end;

if .20 lt s_result le .24 then do;
  nuclide='Cs-136';
end;

if .24 lt s_result le .28 then do;
  nuclide='Cs-137';
end;

if .28 lt s_result le .32 then do;
  nuclide='Gross Alpha';
end;

if .32 lt s_result le .36 then do;
  nuclide='Gross Beta';
end;

if .36 lt s_result le .40 then do;
  nuclide='I-131';
end;

if .40 lt s_result le .44 then do;
  nuclide='I-132';
end;

if .44 lt s_result le .48 then do;
  nuclide='Mo-99';
end;

if .48 lt s_result le .52 then do;
  nuclide='Nb-95';
end;

if .52 lt s_result le .56 then do;
  nuclide='Np-239';
end;

if .56 lt s_result le .60 then do;
  nuclide='Ru-106';
end;

if .60 lt s_result le .64 then do;
  nuclide='Sb-127';
end;

if .64 lt s_result le .68 then do;
  nuclide='Sr-89';
end;

if .68 lt s_result le .72 then do;
  nuclide='Sr-89Ch';
end;

if .72 lt s_result le .76 then do;
  nuclide='Sr-89gamma';
```

```
end;

if .76 lt s_result le .80 then do;
  nuclide='Sr-90';
end;

if .80 lt s_result le .84 then do;
  nuclide='Sr-90Ch';
end;

if .84 lt s_result le .88 then do;
  nuclide='Sr-Total';
end;

if .88 lt s_result le .92 then do;
  nuclide='Te-129m';
end;

if .92 lt s_result le .94 then do;
  nuclide='Te-132';
end;

if .94 lt s_result le .96 then do;
  nuclide='Xe-133';
end;

if .96 lt s_result then do;
  nuclide='Zr-95';
end;
s_id=s_id+1;
output;
end;
end;
end;
run;

proc datasets lib=dblib nolist;
  delete quake8mill_err:
       QUAKE8MILL_ERR:;
  run;
quit;

PROC APPEND BASE=dblib.quake8mil (FASTLOAD=YES BL_LOG=QUAKE8MILL_ERR)
         DATA=work.test;
RUN;
```

APPENDIX 2B: DEMO PROGRAM, QUERY CODE

The code below was written for Teradata. Portions of this code produce expected results without publishing formats to the DBMS, but any sections of the code that use Published optimization (SAS_PUT) function only if formats are published. (See Appendix 1)

```
options nodate nonumber nocenter formdlim='-';

/* database connection information */
%let dbserver=terasoar;
%let dbuser=model;
%let dbpw=&dbuser;
%let database=&dbuser;
```

```
libname dblib teradata server=&dbserver user=&dbuser pw=&dbuser db=&dbuser;

proc format;
  value $isotypeo
  'Ag-110m'='Silver'
  'Am-241'='Americium'
  'Ba-140'='Barium'
  'Ce-144'='Cerium'
  'Cs-134'='Caesium'
  'Cs-136'='Caesium'
  'Cs-137'='Caesium'
  'Gross Alpha'='Gross Alpha'
  'Gross Beta'='Gross Beta'
  'I-131'='Iodine'
  'I-132'='Iodine'
  'Mo-99'='Molybdenum'
  'Nb-95'='Niobium'
  'Np-239'='Neptunium'
  'Ru-106'='Ruthenium'
  'Sb-127'='Antimony'
  'Sr-89'='Strontium'
  'Sr-89Ch'='Strontium'
  'Sr-89gamma'='Strontium'
  'Sr-90'='Strontium'
  'Sr-90Ch'='Strontium'
  'Sr-Total'='Strontium'
  'Te-129m'='Tellurium'
  'Te-132'='Tellurium'
  'Xe-133'='Xenon'
  'Zr-95'='Zirconium'
  other='Other'
  ;

  VALUE rangec
        LOW-0.60  = ' < Min'
        0.60-0.70  ='Trace'
        0.70-0.71 ='Normal  0-10'
        0.71-0.72 ='Normal 10-20'
        0.72-0.73 ='Normal 20-30'
        0.73-0.74 ='Borderline elevated 30-40'
        0.74-0.75 ='Elevated 40-50'
        0.75-0.76 ='Elevated 50-60'
        0.76-0.77 ='Elevated 60-70'
        0.77-0.78 ='Borderline high 70-80'
        0.78-0.79 ='High 80-90'
        0.79-0.80 ='High 90-100'
        0.80-HIGH = 'Alert'
        other   = 'Other';

run;


/* tune system options for "No optimization" */
options sqlgeneration=none
      sqlreduceput=none
      sqlmapputto=none;

%let SYS_SQL_UNPUT_TRACE=1;
%let SYS_SQL_PUT_TRACE=1;
options sql_ip_trace=(note, source) sastrace=',,,d' fullstimer;
```

```
proc sql;
create table all_sas as
select s_id, collection_date,
     nuclide,  put(nuclide, $isotypeo.) as myvar,
     s_result, put(s_result, rangec.) as myvar2
from dblib.quake8mil
where put(nuclide, $isotypeo.)='Strontium' and
     index(left(trim(put(s_result, rangec.))), 'Alert') > 0
order by s_id
;
quit;

options sql_ip_trace=none sastrace=' ' nofullstimer;


/* Tune system options for Local optimization (UNPUT).  Note */
/* the value of sqlgeneration is set to "Yes, generate SQL  */
/* for certain BASE procedures (its default value).  This   */
/* has no effect on format optimization in this case.      */
options sqlreduceput=DBMS
      sqlgeneration=DBMS
      sqlmapputto=none;

%let SYS_SQL_UNPUT_TRACE=1;
%let SYS_SQL_PUT_TRACE=1;

options sql_ip_trace=(note, source)
      sastrace=',,,d' sastraceloc=saslog
      msglevel=i
      fullstimer;

proc sql ;
create table local_optimization as
select s_id, collection_date,
     nuclide,  put(nuclide, $isotypeo.) as myvar,
     s_result, put(s_result, rangec.) as myvar2
from dblib.quake8mil
where put(nuclide, $isotypeo.)='Strontium' and
     index(left(trim(put(s_result, rangec.))), 'Alert') > 0
order by s_id
;
quit;

options sql_ip_trace=none sastrace=' ' nofullstimer;



/* Tune system options to use Published formats (SAS_PUT) */
options sqlgeneration=DBMS
      sqlreduceput=none
      sqlmapputto=SAS_PUT;

%let SYS_SQL_UNPUT_TRACE=1;
%let sys_sql_put_trace=1;


options sql_ip_trace=(note, source) sastrace=',,,d' fullstimer;

proc sql ;
create table published_optimization as
select s_id, collection_date,
     nuclide,  put(nuclide, $isotypeo.) as myvar,
     s_result, put(s_result, rangec.) as myvar2
```

```
from dblib.quake8mil
where put(nuclide, $isotypeo.)='Strontium' and
     index(left(trim(put(s_result, rangec.))), 'Alert') > 0
order by s_id
;
quit;

options sql_ip_trace=none sastrace=' ' nofullstimer;


/* Define a global macro variable containing explicit connection */
/* information to the Teradata server.                           */
%let exconn=%str( server=&dbserver user=&dbuser password=&dbuser
database=&dbuser );

/* Tune system options to use Published formats (SAS_PUT).*/

options sqlgeneration=DBMS
      sqlreduceput=none
      sqlmapputto=SAS_PUT;

%let SYS_SQL_UNPUT_TRACE=1;
%let SYS_SQL_PUT_TRACE=1;


options sastrace=',,,d' sastraceloc=saslog
      sql_ip_trace=(note,source) msglevel=i fullstimer;

proc sql noerrorstop;
 connect to teradata ( &exconn ) ;
 create table explicit_sql as
 select * from connection to teradata (
 select s_id,
      collection_date,
      nuclide,
  sas_put(nuclide,'$ISOTYPEO' ) as myvar,
      s_result,
  sas_put(s_result, 'RANGEC' ) as myvar2
 from "&dbuser."."quake8mil"
 where sas_put(nuclide,'$ISOTYPEO' )='Strontium' and
     sas_put(s_result, 'RANGEC')='Alert'
 order by s_id
 );
quit;

options sastrace=',,,,'
      sql_ip_trace=none msglevel=n nofullstimer;
```

CONTACT INFORMATION

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

David Wiehle
100 SAS Campus Drive Cary, NC 27513
SAS Institute Inc.
david.wiehle@sas.com
http://www.sas.com