Paper 338-2009

Publish SAS® Formats in Your Teradata Server

Jeff Bailey and Pat Bostic, SAS Institute Inc., Cary, NC

ABSTRACT

Using SAS® formats is a great way to simplify your coding and data management needs. Traditionally this has meant extracting the data from your warehouse and processing it locally. This can be very inefficient. Fortunately, the SAS® In-Database Processing initiative, through the SAS and Teradata partnership, has introduced the SAS Formats Library for Teradata. Now SAS formats can be published inside the database, allowing Teradata to do all the work.

This presentation will cover the entire deployment process for SAS formats and custom formats. We will highlight the impact of using these formats in your SAS BI environment.

INTRODUCTION

We have all been there: our SAS code is producing SQL statements that are not passed through to the database. This leads to the unfortunate situation where Teradata passes data to SAS and leaves SAS to postprocess the data. Fortunately, things have been getting better on the implicit pass-through front. We still must be diligent but the situation has gotten much better. On the other hand, there are things that will guarantee that the query sent to the DBMS will be inefficient. Let's take a few moments to discuss one of "those" things.

The SAS PUT function is a SAS programmer favorite. It is often used to implement lookup tables and can have huge, positive impacts on performance. That is, it can have huge positive impacts when used with SAS data sets. However, it has had huge detrimental performance impacts when used with database tables. SAS formats go hand-in-hand with the PUT function. Formats allow SAS programmers to alter the way that data items are presented. Using SAS formats, a programmer can translate one data item to another. For example, you can take a state abbreviation and translate it to the full name of the corresponding state.

Most databases know nothing about the SAS PUT function or SAS formats. For SAS and Teradata users that has changed. The SAS PUT function can now be passed to Teradata. This allows SAS to implicitly pass more efficient SQL statements to Teradata. Implicit pass-through is very important, because it is commonly used in the SAS Business Intelligence environment.

This paper will:

1. Detail the SAS products required to use SAS formats in the Teradata server.
2. Describe, at a high level, the installation process.
3. Describe how to create and use SAS formats inside the Teradata server.

REQUIRED SOFTWARE

Base SAS 9.2 (TS2MO, Rev.920_09w09) and the SAS/ACCESS Interface to Teradata (bundle) is the most current version of the SAS Accelerator Publishing Agent. The software can be used to publish and exploit the SAS formats. This new version of the software is the subject of this paper. Anytime we use the name SAS 9.2 we are referring to this specific version of the SAS 9.2 software.

The previous version of the SAS Accelerator Publishing Agent is SAS 9.2 (TS1M0, Rev .920_08w18+). This version of the software can only publish the formats. It cannot exploit them. We will refer to this version of the software by its official name.

You can exploit the SAS formats using SAS 9.1.3 TS1M3 Service Pack 4 hot fix E9BC46. This version of the software cannot be used to publish the formats. The hot fix names change often: You will definitely need to verify the most current hot fix before trying this at your shop. Anytime we use the name SAS 9.1.3 we are referring to this version of the software.

In order to publish SAS formats inside Teradata you will need to have Base SAS 9.2 (TS2MO, Rev.920_09w09) and the SAS/ACCESS Interface to Teradata (bundle) installed at your site. The SAS/ACCESS Interface to Teradata (bundle) is a software bundle that includes SAS/ACCESS Interface to Teradata, SAS Accelerator Publishing Agent, and SAS Formats Library for Teradata.

The SAS/ACCESS Interface to Teradata (bundle) is available on the following platforms with SAS 9.2 (TS2M0):

- AIX (64-bit enabled)

- HP-UX (64-bit enabled)
- HP-UX IPF
- Linux
- Linux for x64
- Solaris (64-bit enabled)
- Solaris x64
- Windows

You will need a Teradata server in order to publish the SAS formats. That should come as no big surprise. It is important to note that this functionality is not available for every version of Teradata: only Teradata V2R6.2 and Teradata 12.0 on Linux and MP-RAS are supported.

In addition to the Teradata server, you will need the Teradata Tools and Utilities (TTU) installed and configured on the machines where SAS will run. TTU enables client applications, such as SAS, to connect to a Teradata server.

INSTALLATION OVERVIEW

We are not going to get into a detailed installation discussion here. We assume that you currently have a SAS 9.1.3 environment deployed and available for use. You can use this environment to exploit the SAS Formats Library for Teradata.

In order to publish the SAS formats in Teradata, you will need to install Base SAS 9.2 and the SAS/ACCESS Interface to Teradata bundle. That shouldn't be a problem; the installation is very simple. We will refer to the machine where SAS 9.2 is installed as the publishing client.

To enable SAS to communicate with Teradata, you will need to have TTU installed on the publishing client. Your database administrator (DBA) can handle that chore for you.

Here is where things get a little unusual. In order to use the SAS formats in Teradata, your DBA must install the SAS Formats Library for Teradata on your Teradata servers. This software must be installed on each Teradata node. Check the Configuration Guide for SAS 9.2 Foundation for Microsoft Windows for specific details on how to install the SAS Formats Library for Teradata, which will most likely be handled by your DBA.

ARCHITECTURAL OVERVIEW

The SAS Accelerator Publishing Agent takes SAS formats and converts them into C language header and source files. These files are passed to Teradata where they are compiled into Teradata embedded functions. Once the formats have been published to the database, they can be used in SAS programs.

That's all there is to it. The process is very simple as you will see shortly.

PUBLISHING SAS FORMATS IN TERADATA

STEP 1: CONFIGURE THE SAS ENVIRONMENT

We are going to be using custom formats in our environment. You must have the formats defined in the SAS environment and in Teradata. This might seem like an odd requirement, but the SAS SQL parser must have access to the SAS version of the format in order to validate the statement's syntax.

In our example, we want to store these formats in a nondefault location, so we have to configure SAS so that it can find them. To do this we are going to edit the sasv9,cfg file. There is a sasv9,cfg file in the !SASROOT directory. You can make your edits there or in the file referenced by the -CONFIG option.

We need to add the following lines to the sasv9,cfg configuration file:

```
-set fmtlib ("C:\SAS\Production\CustomFormats ")
-fmtsearch (fmtlib work library)
```

The FMTLIB parameter specifies the directory that contains the SAS formats that you publish. This directory can be named anything you wish. For this example we will use the C:\SAS\Production\CustomFormats directory. If you are using both SAS 9.1.3 and SAS 9.2, then the directory that you choose will need to be accessible from both SAS environments. If your networking environments precludes you from doing this, then you can copy the format catalog from one machine to the other.

The -FMTSEARCH option tells SAS where to look for user-defined formats (UDFs). Notice that it references the format library defined by the -SET option.

If you fail to do this step, you will get errors stating that your formats cannot be found.

This step can be omitted if the format catalog is stored in one of the default locations (WORK or LIBRARY). We need to share these libraries between two SAS environments, so using the default locations is not a great idea.

STEP 2: CREATE A SAS FORMAT

In order to publish a SAS format in Teradata we have to create one. To do this we will need to define a SAS library in which to store our SAS format. You can perform this step from either SAS environment. As we mention earlier, if you are using multiple versions of SAS software, then this location must be reachable by both SAS 9.1.3 and SAS 9.2 environments.

In our example we will create the NUMTEXT format and store it in C:\SAS\Production\CustomFormats (remember that we specified this location in the sasv9,cfg file).

We will submit the following code to create our SAS format:

```
LIBNAME prodfmts 'C:\SAS\Production\CustomFormats';
PROC FORMAT LIB=prodfmts;
   VALUE numtext
      1 = 'One'
      2 = 'Two'
      3 = 'Three'
      4 = 'Four';
RUN;
```

You should see no errors in the SAS log when you run the PROC FORMAT code. Make sure that you specified the same directory in the sasv9,cfg file and the LIBNAME statement.

STEP 3: INITIALIZE THE PUBLISHING CLIENT (SAS 9.2)

The first step in the publishing process is to initialize the format publishing environment using the INDTDPF autocall macro. You must do this before you can publish the custom formats in the Teradata server.

You invoke this macro by submitting the following SAS code:

```
%indtdpf;
```

If the initialization was successful you will see the following message in the SAS log:

```
Indtd_publish_formats macro ready for use
```

STEP 4: CONFIGURE CONNECTION INFORMATION (SAS 9.2)

As we mentioned earlier, in order to publish the SAS formats to Teradata we need to connect to the database. The INDCONN macro variable will be used to pass our connection information to the SAS formats publishing client. In order to connect to Teradata we need the following connection information: server, user ID, and password.

You can also specify a database name using this macro. If you omit the database, then the default database for your user ID is used. You can use the database parameter to tell the publishing macro to create the Teradata UDFs in a specific, nondefault database. This is useful if you plan to store your formats in a common location in the database. We will discuss this further in a moment.

You can set the INDCONN macro variable by issuing a SAS statement similar to this one:

```
%let indconn= server="myterasrv" user="myuserid" password="mypassword"
database="myformats";
```

You must assign the INDCONN macro variable before executing the publishing macro. This variable can be set in the autoexec.sas file. If you don't know what the values of these parameters should be in your environment, then check with your DBA.

Let's assume that we have a Teradata server named TERAPROD and a user ID/password (bob/bob1). We want to put the published UDFs in the SASFMT database. Here is the INDCONN macro that will accomplish this:

```
%let indconn= server="TERAPROD" user="bob" password="bob1" database="SASFMT";
```

We will use these connection values throughout the remainder of this paper.

STEP 5: PUBLISH THE FORMAT USING THE PUBLISHING MACRO (SAS 9.2)

Previously, we created a SAS format and stored it in the C:\SASFormats directory. Now, we are going to publish it to our Teradata enterprise data warehouse. We will use the INDTD_PUBLISH_FORMATS macro to do this.

Here is the syntax for the publishing macro (there is no comma before the first argument):

```
%INDTD_PUBLISH_FORMATS (
<DATABASE=database-name>
<,FMTCAT=format-catalog-filename>
<,FMTTABLE=format-table-name>
<,ACTION=CREATE | REPLACE | DROP>
<,MODE=PROTECTED | UNPROTECTED>
<,OUTDIR=diagnostic-output-directory>
);
```

The following is a detailed description of each macro option:

DATABASE — Specifies the name of the Teradata database where the published formats and the format table will be stored. If this argument is omitted, then the formats are published to the user's (specified in the INDCONN macro variable) default database. This argument can be used to store formats in a shared database. This allows multiple users to access them. We will discuss the mechanics of doing this shortly.

You will need the following Teradata privileges on any database where the SAS_PUT functions are stored:

- ALTER FUNCTION
- CREATE FUNCTION
- DROP FUNCTION
- EXECUTE FUNCTION

FMTCAT — Specifies the SAS catalog where our SAS formats are stored. This is always a two-level name. In our example we stored the formats in the C:\SAS\Production\CustomFormats directory. If you create your formats on the SAS 9.1.3 environment, then this directory needs to be reachable via this argument. If you can't reach the directory with both environments, then you can copy the formats to another location. Use this argument to point to the new location. If you omit this argument, then the publishing macro will look search the SAS WORK library for the format.

If you get this value wrong (say, you put in only the SAS library name), it will look like it works but you will encounter errors when you try to use the format.

If you have format definitions in multiple SAS libraries, then you must copy them to a single catalog before publishing. You can use PROC CATALOG to combine multiple SAS catalogs into a single catalog.

FMTTABLE — Specifies the name of a Teradata table that contains a list of all the formats registered by the INDTD_PUBLISH_FORMATS macro. It is a good idea to create this table: it gives the Teradata DBA a way to verify that the custom formats were created during the publishing process.

You can override the DATABASE argument (for the format table location) by prepending the database name to the table name and separating the two with a period. For example, if I want to store this table in the SASINFO database but I have specified DATABASE=SASFMT, then I can use FMTTABLE=SASINFO.SASFORMATS.

You need to have the following privileges on the database where the table will be created in order to use this argument:

- CREATE TABLE
- DROP TABLE

ACTION = CREATE | REPLACE | DROP — This argument controls the creation of the SAS_PUT UDF. Your choices here are CREATE, REPLACE, or DROP. They do exactly what you think that they do. CREATE will create a new function; it will fail if there is already one in the database you specified. REPLACE will create a new function if it already exists. Think of it as a DROP then CREATE. Using REPLACE makes a lot of sense. It is less prone to error. Finally, we have DROP, which deletes the function and does not recreate it. It is useful for cleaning up your environment.

MODE = PROTECTED | UNPROTECTED — Specifies how the SAS_PUT embedded function will execute in Teradata. In PROTECTED mode, embedded functions run outside the Teradata server's address space. This means that they cannot cause the Teradata server to crash. Running in protected mode does not provide the same level of performance as running in UNPROTECTED mode.

In UNPROTECTED mode, embedded functions have the potential of crashing the Teradata server. The SAS functions do not do the "things" that cause Teradata servers to crash. In addition, these embedded functions have been thoroughly tested. Unprotected embedded functions offer the best available performance. We recommend that you use PROTECTED mode for development and testing and then use UNPROTECTED mode in production so that you get the best performance possible. In order to change from PROTECTED mode to UNPROTECTED mode you must republish your formats.

OUTDIR — Specifies a directory where diagnostic information will be written. The files that are created here are very useful when you encounter problems during the publishing process. Let's say you mess up the DATABASE argument: You specified a database that you don't have CREATE TABLE privileges on. You will find out about that here. We can't imagine a situation where you would not want to use this argument.

Let's consider an example. Assume that there are SAS formats stored in C:\SAS\Production\CustomFormats on the publishing client and we need to publish them into the SASFMT database in the TERAPROD Teradata server. Our DBA would like to be able to see the names of the formats that are stored in the database, so we need to create a Teradata table that contains the names. Since this is a production environment we want the best performance available.

Here is what the code looks like:

```
%intdtpf;

LIBNAME prodfmts 'C:\SAS\Production\CustomFormats';

%let indconn= server="TERAPROD" user="bob" password="bob1" database="SASFMT";

%INDTD_PUBLISH_FORMATS(
  FMTCAT=prodfmts.formats
, DATABASE=sasfmt
, FMTTABLE=sasfmt.prodfmt
, ACTION=replace
, MODE=unprotected
, OUTDIR=C:\SAS\Production\PublishOutdir
);
```

At this point, our formats have been successfully published to the Teradata server. Recall that they were written to the location specified by the OUTDIR argument. Many of the errors you will encounter, at least at first, will involve Teradata permission problems. The other common error we have seen is incorrect connection information: specifying the wrong Teradata server name, incorrect user ID, or incorrect password. If you encounter errors during format publishing, you might want to enlist your DBA to help solve them.

You can verify that your SAS_PUT UDF was created by issuing the following SAS code:

```
LIBNAME mytera TERADATA SERVER=teraprod USER=bob PW=bob1 DATABASE=sasfmt;
PROC SQL;
   SELECT fmtname, source, protected
     FROM mytera.prodfmt
    WHERE fmtname = 'NUMTEXT';
QUIT;
```

The SAS formats — including NUMTEXT — have been published to the Teradata server. NUMTEXT is the simple example that we detailed earlier. So, we have our formats published in Teradata; now we need to know how to use them.

STEP 6: USING OUR PUBLISHED CUSTOM SAS FORMATS

Using the formats turns out to be the easy part: using your custom published macros, or any of the standard SAS formats, is as easy as including them in a SAS PUT function. There is something lurking in here that can cause you trouble. Throughout this discussion we have mentioned that the SAS PUT function is actually implemented in Teradata as a UDF named SAS_PUT. We need a way to tell SAS to substitute SAS_PUT when it encounters PUT. This is easy to do: we will use the SYS_SQL_MAPPUTTO SAS macro variable for this purpose.

Here is an example of setting the macro variable:

```
%LET SYS_SQL_MAPPUTTO=sas_put;
```

It is possible to use a name other than SAS_PUT for the Teradata embedded functions. You might want to do this if you are storing the functions for multiple groups in one location. If you want to make these functions available to all users, you can store them in the Teradata SYSLIB database. This will make the functions available to all users. Keep in mind that you will need to develop naming conditions so that you don't overwrite functions used by other SAS users.

In our example we have stored our Teradata UDF in a nondefault database. This is the most likely situation you will encounter. We have to do something special to handle this. We need to preface SAS_PUT with the database name. In our example we are publishing the function to the PRODFMTS database. Here is the correct macro definition for this situation:

```
%LET SYS_SQL_MAPPUTTO=prodfmts.sas_put;
```

Let's say that we want to use the NUMTEXT format in a query. We will need a Teradata table to use in our example. Fortunately, we are creating a very simple one using a SAS DATA step. Notice that we are defining our numeric column as an integer in the database. In addition, we are explicitly setting the primary index for the table. Using the format published in Step 5 as an example, here is what the code could look like:

```
LIBNAME mytera TERADATA SERVER=TERAPROD USER=bob PW=bob1;

DATA mytera.numbers (dbtype=(mynum=int)
                     dbcreate_table_opts='PRIMARY INDEX(mynum)');
   mynum = 4; output; mynum=3; output; mynum=2; output; mynum=1; output;
   mynum = 4; output; mynum=3; output; mynum=2; output;
   mynum = 4; output; mynum=3; output;
   mynum = 4; output;
RUN;

OPTIONS SASTRACE=',,,d' SASTRACELOC=saslog NOSTSUFFIX;

%LET SYS_SQL_MAPPUTTO=prodfmts.sas_put;

PROC SQL;
   CREATE TABLE work.number_freqs AS
      SELECT PUT(mynum, NUMTEXT.) as Number
          , COUNT(mynum) AS Number_Cnt
        FROM mytera.numbers
       GROUP BY mynum;
QUIT;
```

Setting the SASTRACE= and SASTRACELOC= options allows us to see the actual SQL that SAS is passing to Teradata. The NOSTSUFFIX option limits the tracing output to only that which is necessary. In the SAS log we should see that the PUT function was translated into the PRODFMTS.SAS_PUT Teradata UDF.

Here is the resulting SAS log:

```
NOTE: Copyright (c) 2002-2003 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) 9.1 (TS1M3)
      Licensed to Employee SID, Site 0055438001.
NOTE: This session is executing on the XP_PRO  platform.



NOTE: SAS 9.1.3 Service Pack 4

NOTE: SAS initialization used:
      real time           1.17 seconds
      cpu time            0.60 seconds

1     LIBNAME teraprod TERADATA SERVER=teraprod USER=bob PW=XXXX;
```

```
NOTE: Libref TERAPROD was successfully assigned as follows:
      Engine:        TERADATA
      Physical Name: teraprod
2
3    OPTIONS SASTRACE=',,,d' SASTRACELOC=saslog NOSTSUFFIX;
4
5    %LET SYS_SQL_MAPPUTTO=SASFMT.SAS_PUT;
6
7    PROC SQL;
8       CREATE TABLE work.number_freqs AS
9          SELECT PUT(mynum, NUMTEXT.) as Number
10              , COUNT(mynum) AS Number_Cnt
11           FROM teraprod.numbers
12          GROUP BY mynum;

TERADATA_0: Prepared:
SELECT * FROM "numbers"


TERADATA_1: Prepared:
 select cast(SASFMT.SAS_PUT("numbers"."mynum", 'NUMTEXT5.0') as char(5)) as
"Number",
COUNT("numbers"."mynum") as "Number_Cnt" from "numbers" group by "numbers"."mynum"

ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.

TERADATA_2: Executed:
 select cast(SASFMT.SAS_PUT("numbers"."mynum", 'NUMTEXT5.0') as char(5)) as
"Number",
COUNT("numbers"."mynum") as "Number_Cnt" from "numbers" group by "numbers"."mynum"

TERADATA: trget - rows to fetch: 4
NOTE: Table WORK.NUMBER_FREQS created, with 4 rows and 2 columns.

13   QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.31 seconds
      cpu time            0.04 seconds


14
15   PROC PRINT DATA=work.number_freqs;
16   RUN;

NOTE: There were 4 observations read from the data set WORK.NUMBER_FREQS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.42 seconds
      cpu time            0.01 seconds
```

Look at the TERADATA_2 section of the log. It is clear that SASFMT.SAS_PUT was substituted for PUT. There were four observations written to the WORK.NUMBER_FREQS SAS data set. That is a good sign that things worked well. In order to prove that it worked, we will need to examine the data:

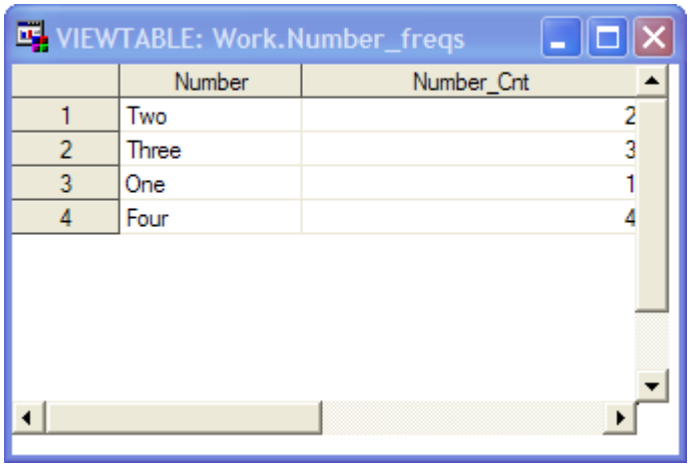

Figure 1. Results from the Format Run

This is exactly what we expect to see. Our custom format was applied and the integer values were converted to text.

TROUBLESHOOTING

Let's consider some of the most common problems that you might encounter and discuss how you can fix them.

ERROR: The format <format-name> was not found or could not be loaded.

When you see this error you can assume that your format, for some reason, cannot be found. You will want to make sure that you have added the -SET and -FMTSEARCH options to the sasv9,cfg files on both your SAS 9.1.3 and

SAS 9.2 environments.

We have also seen this error when the format name is spelled incorrectly.

ERROR: Teradata row not delivered (trget): in UDF/XSP/UDM <database-name>.sasputn: SQLSTATE 22023.

The first thing that you want to do is verify that your custom format was published to Teradata. You can do this by querying the Teradata table containing the formats list: You did create that, right? Recall that this table is created when you pass the FMTTABLE argument to the publishing macro. Chances are that you will not find your custom format listed in the table.

This code will search the Teradata table specified in FMTTABLE for the format we created earlier:

```
LIBNAME mytera TERADATA SERVER=TERAPROD USER=bob PW=bob1;
PROC SQL;
   select fmtname, source, protected
     FROM mytera.prodfmt (DATABASE=SASFMT)
    where fmtname = 'NUMTEXT';
QUIT;
```

We have seen this error when we provided an incorrect value for the FMTCAT argument when invoking the publishing macro. The FMTCAT takes a SAS library and a catalog name. Once, we specified only the library name and got this error.

CONCLUSION

Publishing SAS formats into your Teradata enterprise data warehouse can greatly increase the performance of your SAS code. It gives programmers greater flexibility in designing their SAS programs. Using SAS formats inside the Teradata server allows SAS programmers to be SAS programmers.

We are sure that your head is spinning right now. We have covered a great deal of information. You might be contemplating your next step. Here is what we want you to do: Give some serious thought as to how you can take

what you have learned back to your shop and put it to use. Then contact your SAS representative and find out how you can get the software.

REFERENCES

SAS Institute Inc., 2009. "Deploying and Using SAS Formats in Teradata." In *SAS/ACCESS 9.2 for Relational Databases: Reference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/cdl/en/acreldb/61890/HTML/default/a003276900.htm.

SAS Institute Inc., 2009. *Configuration Guide for SAS 9.2 Foundation for Microsoft WIndows*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/installcenter/en/ikfdtnwincg/62175/PDF/default/config.pdf.

ACKNOWLEDGMENTS

The authors extend their heartfelt thanks and gratitude to the following individuals:

Pravin Boniface, SAS Institute

Brian Mitchell, Teradata

Phil Mohr, SAS Institute

Greg Otto, Teradata

Austin Swift, SAS Institute

CONTACT INFORMATION

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

Jeff Bailey
SAS Institute
SAS Campus Drive, T6118
Cary, NC 27513
(919) 531-6675
Jeff.Bailey@sas.com

Pat Bostic
SAS Institute
SAS Campus Drive, R3238
Cary, NC 27513
(919) 531-7949
Pat.Bostic@sas.com