Paper 3314-2015

A Metadata to Physical Table Comparison Tool

David Moors, Whitehound Limited, UK

ABSTRACT

When promoting metadata in large packages between environments, metadata and the underlying physical data can become out of sync. This can result in metadata items that cannot be opened because SAS® clients have thrown an error. It often falls to the SAS administrator to resolve the synchronization issues when they might not have even been responsible for promoting the metadata items in the first place.

In this paper, we will discuss a simple program that can be used to compare the table metadata to that of the physical tables, and any anomalies will be noted.

METADATA

With the introduction of the SAS 9 platform in 2002, SAS introduced the concept of metadata as a way of providing a centralised data layer between all SAS Applications. The official definition of the metadata server taken from the SAS website is as follows:

*The **SAS Metadata Server** is a centralized resource for storing, managing, and delivering metadata for SAS applications across the enterprise. The SAS Metadata Server enables centralized control so that all users access consistent and accurate data. Access to data and metadata is secured through a metadata-based authorization layer, which supplements protections from the host environment and other systems. The functionality of the SAS Metadata Server is provided through the **SAS Open Metadata Architecture**, which is a metadata management facility that provides common metadata services to applications.*

It is the last paragraph from the official definition, relating to the SAS Open Metadata Architecture, which is fundamental in providing the ability to query metadata programmatically and is central to the concepts in this paper.

THE ISSUE

The use of metadata allows the ability to separate of SAS environments into separate logical levels. In the early days of SAS 9.1, this may have been 3 levels for Development (Lev1), Test (Lev2) and Production (Lev3). However, now it is not uncommon to have anywhere between 5 to 8 Levels of metadata.

In nearly all cases it is good practise to develop your code and programs in the Development environment (Lev1) and promote the package up through the various Levels in your SAS Environments.

Usually the packaging of metadata tasks falls to the SAS Administrators, however depending on security policies, time, budget or skillsets this can fall to other individuals in the organisation who may be less au fait about data synchronisation between SAS environments.

One of more common problems that can arise when promoting metadata between environments is that the metadata and physical data can become out-of-sync.

Prior to the release of Metadata-Bound libraries in SAS 9.3 M2, anyone with the appropriate SAS clients e.g. SAS Enterprise Guide, SAS Display Manager were able assign libnames direct to physical SAS datasets and manipulate the underlying data. SAS Metadata-Bound libraries are now able to force SAS clients to first visit the metadata server, whereby the metadata security policy can determine what data users can access.

The issue of out-of-sync metadata and physical data can be a common one, and can lead to user frustration and lack of confidence in the metadata platform, to the point whereby users only assign physical libraries to deal with data, which is not a desirable outcome. Figure 1 below shows a typical error that SAS Enterprise Guide can throw when attempting to open a dataset assigned with the Meta engine when the physical and metadata tables are out-of-sync.

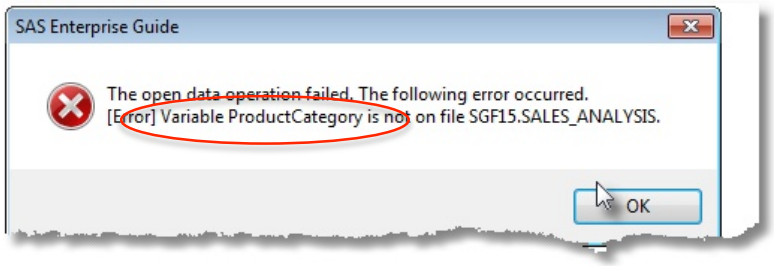
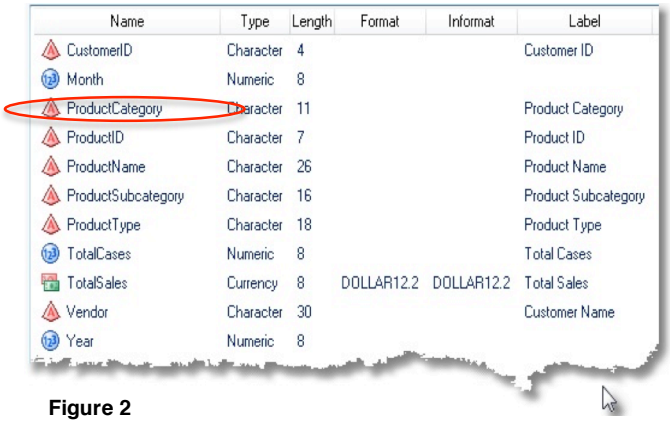


Figure 1

Above we can see that SAS Enterprise Guise is indicating that the variable 'ProductCategory' is not contained in the SAS dataset 'SGF15.SALES_ANALYSIS'.

However if we look at the variables in the 'Properties' of dataset 'SGF15.SALES_ANALYSIS' (assigned using the using the META engine in SAS Enterprise Guide), we can see that the variable 'ProductCategory' exists in table Metadata. This can be seen in Figure 2, opposite.



Name	Type	Length	Format	Informat	Label
CustomerID	Character	4			Customer ID
Month	Numeric	8			
ProductCategory	Character	11			Product Category
ProductID	Character	7			Product ID
ProductName	Character	26			Product Name
ProductSubcategory	Character	16			Product Subcategory
ProductType	Character	18			Product Type
TotalCases	Numeric	8			Total Cases
TotalSales	Currency	8	DOLLAR12.2	DOLLAR12.2	Total Sales
Vendor	Character	30			Customer Name
Year	Numeric	8			

Figure 2

Whilst the message (Figure 1, above) from SAS Enterprise Guide is true, in that the variable 'ProductCategory' is missing from the physical dataset, SAS Enterprise Guide may not be providing us with the full picture in relation to any further missing variables.

If we were to open the 'SGF15.SALES_ANALYSIS' table in SAS Data Integration Studio the following Warning, shown in Figure 3 below, would be displayed.

In the case of Figure 3, SAS DI Studio is advising us that columns/variables ProductCategory, TotalCases and ProductType are missing from the SGF15.SALES_ANALYSIS table.

A curious user could then proceed to open the 'SGF15.SALES_ANALYSIS' table in SAS Display Manager (AKA PC SAS) and see the physical table (dataset) has the following variables. The output can be seen in Figure 4, below.

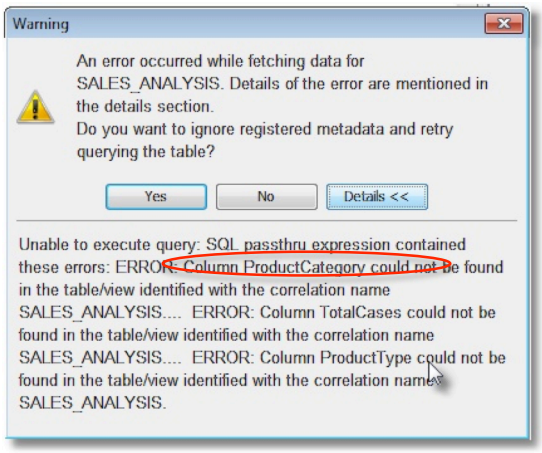
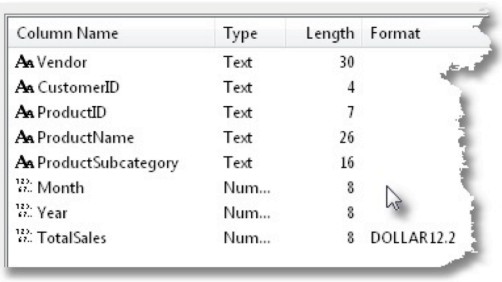


Figure 3



Column Name	Type	Length	Format
Vendor	Text	30	
CustomerID	Text	4	
ProductID	Text	7	
ProductName	Text	26	
ProductSubcategory	Text	16	
Month	Num...	8	
Year	Num...	8	
TotalSales	Num...	8	DOLLAR12.2

Figure 4

By comparing metadata output from SAS Enterprise Guide (Figure 2) and the physical data from SAS Display Manager (Figure 4) we can see that three variables are indeed different between the metadata and physical data. This was correctly identified by the Warning when opening the table in SAS DI Studio (Figure 3), but sadly not by SAS Enterprise Guide (Figure 1).

As, it could be argued, that the majority of users accessing data will arrive via SAS Enterprise Guide, and as SAS Enterprise Guide is SAS' recommended client of choice for the SAS 9 platform, this is obviously an issue.

This lends the question of "what can be done to resolve this common issue"?

THE RESOLUTION – MetaToPhysicalTableComparisonTool (MTPTCT)

Fortunately the Meta2PhysicalTableComparisonTool (MTPTCT) can assist in resolution of out-of-sync metadata and physical data.

At the simplest level the MetaToPhysicalTableComparisonTool takes two inputs: a metadata table and physical dataset and outputs the differences between the two data structures.

The following section will detail the parts that go into making up the MTPTCT and how this can be used practically in a real world scenario.

The MetaToPhysicalTableComparisonTool is comprised of the following parts, each of which will be described in more detail:

- Two libname inputs
- A SAS DATA Step to query the Metadata server
- PROC CONTENTS for the physical table
- PROC COMPARE on the outputs from Metadata server and PROC CONTENTS

Libnames

In order to be able to provide a comparison between table data held within the SAS Metadata Server and physical SAS datasets, both a META engine libname and a Physical libname will need to assigned to point to the table in which you wish to compare.

```
/* assign METADATA library */
libname SGF15 meta library="SGF15";

/* assign PHYSICAL library */
libname sgf15p 'S:\SGF_Paper_Files\2015\data';
```

Figure 5

In the case of Figure 5 above, the META engine library is being assigned to 'SGF15', and the Physical library is being assigned to 'sgf15p'.

The MetaToPhysicalTableComparisonTool is macro parameter based and will accept the following parameters:

MtableName – this is the name of the table to search in Metadata
plib – this is the physical libname assigned
PtableName – this is the physical table name.

Note: A metadata library paramater is not required as the MtableName value will be issued to search for the table on the Metadata server by its name. More will be discussed on this subject later on in this section.

The Macro code will query the value of the macro parameter &PtableName. If the macro parameter for &PtableName is set to missing, then the value of &MtableName (the metadata table name) is assigned to &PtableName (the physical table name).

The assumption for the logic above is that it is good development practice to provide your metadata table names with that of the physical table names or vice versa. Thus, unless otherwise stated, the metadata and physical tables will use the same name for all further calls in the macro code block. The code for this logic can be seen in Figure 6 below:

```
%macro meta_compare(MtableName=, plib=, PtableName=);

   /* check Physical table name is empty. Assign metadata name to physical
       name macro variable */
   %if &PtableName = %then %do;
      %let PtableName = &MtableName;
      %put Physical Table Name is now called: >>> &PtableName >>>;
   %end;
```

Figure 6

SAS DATA Step– Query Metadata

When looking to access metadata programmatically, SAS provides the 'SAS Language Interface to Metadata' as a way in which to access the obtain information contained in the Metadata Server.

Common ways in which you can access metadata include: Metadata LIBNAME statements, PROC METALIB, PROC METADATA and the SAS DATA Step. It is the later that will to be used to programmatically query the metadata server.

As the purpose of the MTPTCT is to compare the physical to metadata table properties it would be prudent to select some table properties to compare. As the most common cause for physical and metadata tables being out-of-sync is missing columns, this is the area on which the MTPTCT has been focused.

Before using some of the Functions provided by the 'SAS Language Interface to Metadata' for querying metadata (described later on in this section), the column variables in the DATA set needed to be defined. This was achieved using the LENGTH statement, as can be seen in Figure 7 below.

```
/*-----------------------------------------------------*/
/* obtain metadata view of the columns in the dataset */
/*-----------------------------------------------------*/
%let metaDSN=work.metadata_columns;

 data &metaDSN(keep=column_name format_type    column_length);
     length column_name $32.
            column_length $6.
            format_type $200.
            columnURI tableURI metatype $256.
            ;
```

Figure 7

It is worth noting that all the variables have been defined as character, as this is the data type returned when querying the metadata server.

As mentioned earlier 'The SAS Language Interface to Metadata' provided numerous Functions that can be used in the DATA Step to query metadata. The most commonly used ones can be seen in Figure 8 below.

Function Syntax	Description
METADATA_RESOLVE(uri, type, id)	Resolves a metadata URI into a specific object type
METADATA_GETATTR(uri, attr, value)	Returns the named attribute for the object specified by the URI
METADATA_GETNASL(uri, n, asn)	Returns the nth named association for the object URI
METADATA_GETNASN(uri, asn, n, nuri)	Returns the nth associated object of the association specified
METADATA_GETNATR(uri, n, attr, value)	Returns the nth attribute on the object specified by the URI
METADATA_GETNOBJ(uri, n, nuri)	Returns the nth object matching the specified URI
METADATA_GETNPRP(uri, n, prop, value)	Returns the nth property on the object specified by the input URI
METADATA_GETNTYP(n, type)	Returns the nth object type on the server
METADATA_GETPROP(uri, prop, value)	Returns the named property for the object specified by the input URI

Figure 8

For the purposes of the MTPTCT only three of these function were required. **METADATA_RESOLVE**, **METADATA_GETNASN** and **METADATA_GETATTR**.

To query the metadata server to find a table name we use the **METADATA_RESOLVE** Function with the object type of 'PhysicalTable'. The name of the table being queried is passed to the Function in the macro variable &MtableName (in the case of this example "SALES_ANALYSIS") and the value returned from the Function is stored in the variable 'tableURI'.

If the table has been successfully located in the metadata server, the 'tableURI' variable will be populated with the metadata identifier for that object. For the 'SALES_ANALYSIS' table this would be: A5NNUJQ0.BJ0000SG. Here:

- The first eight characters (A5NNUJQ0) identify the SAS Metadata Repository in which the object is stored.
- The ninth character is always a fullstop (period)
- The second of the eight character (BJ0000SG) identifies the object in the repository

Next the **METADATA_GETNASN** function is used to return the number of columns within the table. Here the tableURI variable (A5NNUJQ0.BJ0000SG) from the previous statement is used to get the columns objects associated with the table. The values are then passed into the variable 'numCols' which will hold the number of columns in the metadata table. The code showing the usage of the metadata function **METADATA_RESOLVE** and **METADATA_GETNASN** can be seen in Figure 9 below.

```
/* The CALL MISSING statement initialises the output variables to missing
    values. */
call missing (of _all_);

/* The METADATA_RESOLVE function gets PhysicalTable objects associated with the
    table name and passed the metadata identifier to the variable 'tableURI'
*/

rc=metadata_resolve("omsobj:PhysicalTable?@Name='&MtableName'",metatype,tableURI
);

/* The METADATA_GETNASN function gets objects associated via the table URI.
*/
numCols=metadata_getnasn(tableURI,"Columns",1,columnURI);
```

Figure 9

In order to return all the columns associated with a metadata table a DO-Loop is required to iterate through all the columns. The code in Figure 10 below illustrates this concept.

```
/* The DO statement specifies a group of statements to be executed as a unit.

The METADATA_GETATTR function gets the values of the Column Name, Column Length
and SAS Format attributes. */
   do colnum=1 to numCols;
       numcols=metadata_getnasn(tableURI,"Columns",colnum,columnURI);
       rc=metadata_getattr(columnURI,"SASColumnName",column_name);
       rc=metadata_getattr(columnURI,"SASColumnLength",column_length);
       rc=metadata_getattr(columnURI,"SASFormat",format_type);
       output;
   end;
run;
```

Figure 10

In the code above the variable ‘colnum’ is assigned a value of 1 and the loop will iterate through to the value contained in the variable numCols, which was assigned the value from the **METADATA_GETNASN** function described above.

The loop contains the metadata function **METADATA_GETATTR,** which takes the metadata identifier associated with the ‘columnURI’ variable and returns the attribute associated with the metadata identifier. For the purposes of the MTPTCT the following attributes were requested and put into the corresponding variable names:

SASColumnName → column_name
SASColumnLength → column_length
SASFormat →format_type

In order to stop only the final values from the last iteration of the do-loop solely being written to the SAS dataset, an OUTPUT statement was used to retain all values for the variables defined in the loop.

If we were not to exclude any of the variables defined in the MTPTCT code described above we would end up with a DATA set as per Figure 11 below:

	column_name	column_length	format_type	columnURI	tableURI	metatype	numcols	colnum
1	Vendor	30		OMSOBJ:Column\A5NNUJQ0.BL00014L	A5NNUJQ0.BJ0000SG	PhysicalTable	11	1
2	CustomerID	4		OMSOBJ:Column\A5NNUJQ0.BL00014M	A5NNUJQ0.BJ0000SG	PhysicalTable	11	2
3	ProductID	7		OMSOBJ:Column\A5NNUJQ0.BL00014N	A5NNUJQ0.BJ0000SG	PhysicalTable	11	3
4	ProductName	26		OMSOBJ:Column\A5NNUJQ0.BL00014O	A5NNUJQ0.BJ0000SG	PhysicalTable	11	4
5	ProductCategory	11		OMSOBJ:Column\A5NNUJQ0.BL00014P	A5NNUJQ0.BJ0000SG	PhysicalTable	11	5
6	ProductSubcateg...	16		OMSOBJ:Column\A5NNUJQ0.BL00014Q	A5NNUJQ0.BJ0000SG	PhysicalTable	11	6
7	Month	8		OMSOBJ:Column\A5NNUJQ0.BL00014R	A5NNUJQ0.BJ0000SG	PhysicalTable	11	7
8	Year	8		OMSOBJ:Column\A5NNUJQ0.BL00014S	A5NNUJQ0.BJ0000SG	PhysicalTable	11	8
9	TotalCases	8		OMSOBJ:Column\A5NNUJQ0.BL00014T	A5NNUJQ0.BJ0000SG	PhysicalTable	11	9
10	TotalSales	8	DOLLAR12.2	OMSOBJ:Column\A5NNUJQ0.BL00014U	A5NNUJQ0.BJ0000SG	PhysicalTable	11	10
11	ProductType	18		OMSOBJ:Column\A5NNUJQ0.BL00014V	A5NNUJQ0.BJ0000SG	PhysicalTable	11	11

Figure 11

In the figure above we can see that ‘columnURI’ and ‘tableURI’ have returned their metadata identities, and the variables: column_name, column_length and format_type have been populate with the value held on the metadata server.

However, for the purposes of the MTPTCT, not all of the variable seen in Figure 11 above, are required. Thus the KEEP dataset option was used (seen Figure 7) to retain only the variables required.

The final table output from querying the metadata server, required as part of the MTPTCT, can be seen in Figure 12, opposite:

METADATA_COLUMNS

Filter and Sort | Query Builder | Data | Describe | Graph | An...

	column_name	column_length	format_type
1	CUSTOMERID	4	
2	MONTH	8	
3	PRODUCTCATE...	11	
4	PRODUCTID	7	
5	PRODUCTNAME	26	
6	PRODUCTSUBC...	16	
7	PRODUCTTYPE	18	
8	TOTALCASES	8	
9	TOTALSALES	8	DOLLAR12.2
10	VENDOR	30	
11	YEAR	8	

Figure 12

PROC CONTENTS

Now that the relevant table attributes have been returned from the metadata server and put into a SAS dataset, a similar process is required for the physical table in order to capture the data for the comparison.

Luckily the physical table attributes for a table can be easily accessed using PROC CONTENTS.

The code to extract the information required can be seen in Figure 13 below.

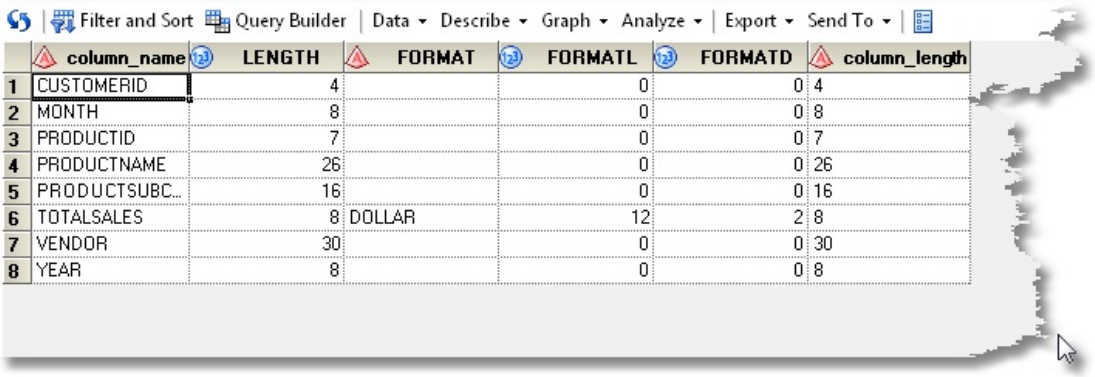
```
/*---------------------------------------*/
/* export physical columns to a dataset */
/*---------------------------------------*/
   %let physicalDSN=work.physical_columns;

   proc contents data = &plib..&PtableName noprint
                 out = &physicalDSN(rename=(name=column_name)
                                    keep=name length format formatl formatd
                                    );
   run;
```

Figure 13

Here the libname and table name returned from the macro parameters (described earlier), are used on the DATA option. The OUT option is then used to write out a physical table.

Due to the verbose output returned from using PROC CONTENTS a KEEP option was used to retain only the values required from the table.

The output from the PROC CONTENTS procedure can be seen in Figure 14 below:



	column_name	LENGTH	FORMAT	FORMATL	FORMATD	column_length
1	CUSTOMERID	4		0	0	4
2	MONTH	8		0	0	8
3	PRODUCTID	7		0	0	7
4	PRODUCTNAME	26		0	0	26
5	PRODUCTSUBC...	16		0	0	16
6	TOTALSALES	8	DOLLAR	12	2	8
7	VENDOR	30		0	0	30
8	YEAR	8		0	0	8

Figure 14

In order to get the data into a state that resembled the output from querying the metadata server, an addition DATA Step was required. This additional step is required as PROC CONTENTS returns format information over three variables. These are:

FORMAT → The type of format
FORMATL → The length of the format
FORMATD →The format decimal precision.

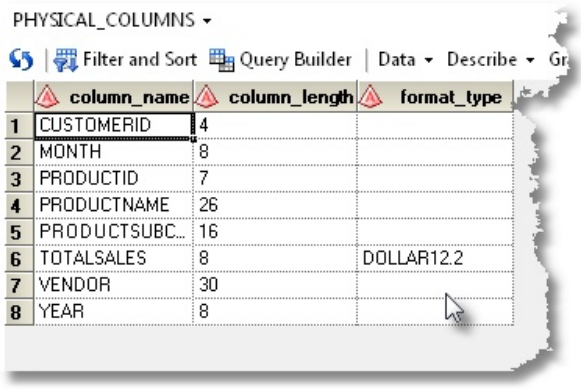
This is in contrast to the format information returned from the metadata server, which contains all elements of the format in one string.

The DATA Step code used to align the data from PROC CONTENTS to that returned from the metadata server can be seen in Figure 15 below:

```
/* convert column names to uppercase */
data &physicalDSN(keep= column_name column_length format_type);
    set &physicalDSN;

    column_name=upcase(column_name);
    column_length=put(strip(length),$6.);

    if formatl NE 0 and formatd > 0 then
         format_type=cats(format,formatl,'.',formatd);
    else if formatl NE 0 then format_type=cats(format,formatl,'.');
    else if formatl = 0 and formatd = 0 and format NE '' then
         format_type=cats(format,'.');
run;

/* sort variables into alphabetical order */
proc sort data=&physicalDSN;
    by column_name;
run;
```

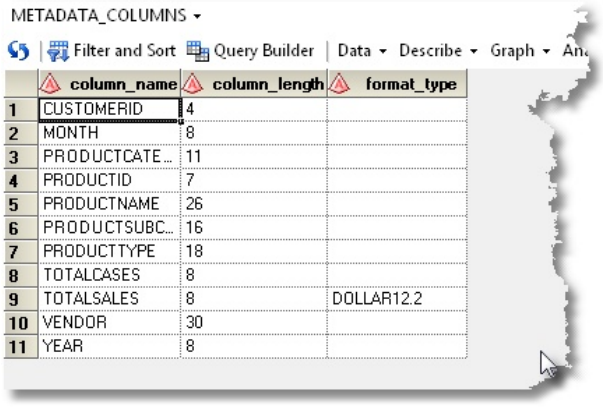
Figure 15

The final physical table can be seen in Figure 16 below, along with a side-by-side comparison to the dataset created from querying the metadata server (Figure 17).



	column_name	column_length	format_type
1	CUSTOMERID	4	
2	MONTH	8	
3	PRODUCTID	7	
4	PRODUCTNAME	26	
5	PRODUCTSUBC...	16	
6	TOTALSALES	8	DOLLAR12.2
7	VENDOR	30	
8	YEAR	8	

Figure 16



	column_name	column_length	format_type
1	CUSTOMERID	4	
2	MONTH	8	
3	PRODUCTCATE...	11	
4	PRODUCTID	7	
5	PRODUCTNAME	26	
6	PRODUCTSUBC...	16	
7	PRODUCTTYPE	18	
8	TOTALCASES	8	
9	TOTALSALES	8	DOLLAR12.2
10	VENDOR	30	
11	YEAR	8	

Figure 17

Comparing the output:

Now that we have the two data extracts, one from querying the metadata server and one from querying the physical table, the two tables can be compared to check for any differences. Quickly looking at Figures 16 and Figure 17 above it can be clearly seen that the two datasets don't match.

However, for demonstration purposes the 'SALES_ANALYSIS' table doesn't contain many variables thus making it easy see any issues, but for datasets with significantly more variables eyeballing data can be impractical and time consuming. Therefore the best way to compare two datasets is using PROC COMPARE.

For the purposes of what is required by the MTPTCT, only the basic syntax and default output from PROC COMPARE is required. The code used to compare the two datasets can be seen in Figure 18 below.

```
/*-----------------------------------------------------------------*/
/* compare metadata table attributes to physical c table attributes */
/*-----------------------------------------------------------------*/
proc compare base=&physicalDSN
             compare=&metaDSN
             ;
run;
```

Figure 18

RESULTS

From the output provided by PROC COMPARE the results shown in Figure 19 below we can see that there are three additional columns in the metadata data version of the SALES_ANALYSIS table compared to the physical table. These results can be confirmed by referring back to Figures 16 and 17 earlier in the paper when a visual side-by-side analysis was performed.

```
                          The COMPARE Procedure
        Comparison of WORK.PHYSICAL_COLUMNS with WORK.METADATA_COLUMNS
                                (Method=EXACT)

                            Data Set Summary

Dataset                          Created          Modified  NVar    NObs

WORK.PHYSICAL_COLUMNS  04FEB15:20:59:00  04FEB15:20:59:00     3       8
WORK.METADATA_COLUMNS  04FEB15:20:59:00  04FEB15:20:59:00     3      11

             Listing of Common Variables with Differing Attributes

      Variable     Dataset                Type  Length  Label

      column_name  WORK.PHYSICAL_COLUMNS  Char      32  Variable Name
                   WORK.METADATA_COLUMNS  Char      32


                             Observation Summary

                         Observation      Base  Compare

                         First Obs           1        1
                         First Unequal       3        3
                         Last  Unequal       8        8
                         Last  Match         8        8
                         Last  Obs           .       11

Number of Observations in Common: 8.
Number of Observations in WORK.METADATA_COLUMNS but not in WORK.PHYSICAL_COLUMNS: 3.
Total Number of Observations Read from WORK.PHYSICAL_COLUMNS: 8.
Total Number of Observations Read from WORK.METADATA_COLUMNS: 11.
```

Figure 19

From the results seen in Figure 19, it can be confirmed that some action needs to be performed on either the metadata or physical table to get the tables back into sync, and avoid such errors that the SAS Clients are throwing, as seen in Figures 1 and 3 earlier in this paper.

Use case: Practical example

Whilst looking at a single table for disparities between the metadata and physical attributes can be useful for a SAS Administrator, this process would need to be scaled up to query all tables for a given physical libname, or multiple physical libnames.

Fortunately this can be easily achieved using PROC DATASETS, PROC SORT and a null DATA Step using CALL EXCUTE and the MTPTCT macro. The code for this process can be seen in Figure 20 below.

```
/*-------------------------------------------------*/
/* Export all dataset attributes for SGF15P library */
/*-------------------------------------------------*/
proc datasets library=SGF15P memtype=data nodetails nolist;
    contents out=work.listDatasets (keep=libname memname)
    data=_all_
    noprint;
quit;


/*---------------------------------------*/
/* remove duplicate dataset name entries */
/*---------------------------------------*/
proc sort data=work.listDatasets nodupkey;
    by memname;
run;


/*-------------------------------------------------------------*/
/* execute the 'meta_compare' macro for all datasets in SGF15P */
/*-------------------------------------------------------------*/
data _null_;
    set work.listDatasets;
    call execute('%meta_compare(MtableName='||strip(memname)||',
                    plib='||libname||', PtableName=);');
run;
```

Figure 20

In the above code the CONTENTS and the OUT options are used in PROC DATASETS to output the libname and the memname (dataset name), for all the datasets held in the SGF15P library, to a dataset called ‘listDatasets’.

A PROC SORT is then used on the ‘listDatasets’ dataset to remove duplicate memnames (dataset names) as the output from PROC DATASETS contains all variables in every dataset in the library, thus the memname is duplicated multiple times for every dataset.

Finally a null DATA Step is used to run a CALL EXECUTE statement for every dataset contained in the memname variable in the 'listDatasets' table. The DATAset variables 'memname' and 'libname' are used to pass the values in the dataset into the macro variable parameters 'MtableName' and 'plib'.

Using the above method, the MTPTCT can be run on entire libraries of data.

One last thing…

A significant amount of the out-of-sync metadata and physical data issues are caused by importing metadata between SAS environments, especially when using SAS Data Integration Studio.

There's a relatively unknown Java argument that can be added to the 'distudtio.ini' file that can opens up a few additional options in SAS Data Integration Studio when importing metadata.

The options is as follows:

JavaArgs_15=-**Dcom.sas.metadata.logical.table.TableOptionsPanel.MergeOptionVisible=TRUE**

Note: the JavaArgs should be one higher than the already highest value assigned in the distudio.ini file.

Figure 21 below shows the options added to my distudio.ini file.

```
[properties]
MASTERPROP="C:\Program Files\SAS\sassw.config"

[default]
startdir=<LAUNCHERDIR>
applogloc=
launchercmd=<JREHOME>
JavaArgs_1=-Xmx1024m
JavaArgs_2=-XX:MaxPermSize=128m
JavaArgs_3=-Djava.security.auth.login.config="security\login.config"
JavaArgs_4=-Djava.security.policy="security\auth.policy"
JavaArgs_5=-DentityExpansionLimit=1000000
JavaArgs_6=-Dsas.app.splash.path=splash.gif
JavaArgs_7=-Dlog4j.configuration="file:\C:\Program Files\SAS\SASDataIntegrationStudio\4.3\log4j.properties"
JavaArgs_8=-Dsas.services.information.types.path="C:\Program Files\SAS\SASPlatformObjectFramework\9.3\plugins"
JavaArgs_9=-Djava.system.class.loader=com.sas.app.AppClassLoader
JavaArgs_10=-Dsas.ext.config="<SASHOME>\sas.java.ext.config"
JavaArgs_11=-Dsas.app.launch.config=picklist
JavaArgs_12=-Dsas.app.repository.path="<VJRHOME>\eclipse"
JavaArgs_13=-Dsas.app.class.path=.;"C:\Program Files\SAS\SASDataIntegrationStudio\4.3\build"
JavaArgs_14=-Dsas.app.launch.picklist=picklist;"help\primary.picklist"
JavaArgs_15=-Dcom.sas.metadata.logical.table.TableOptionsPanel.MergeOptionVisible=TRUE
MainClass=com.sas.wadmin.application.TheAppWA
Classpath=-cp "<VJRHOME>/eclipse/plugins/sas.launcher.jar"
```

Figure 21

The result from adding this option to the distudio.ini file is that when it comes to importing metadata tables two additional check boxes are added to the 'Options' tab, and a 'Change Analysis' button is made available.

The two check boxes provide the following options:

- Keep target columns not found in the source
- Include source columns not found in the target

The 'Change Analysis' button opens up a new window in SAS Data Integration Studio showing the impact the combination from the check boxes have when importing the table metadata.

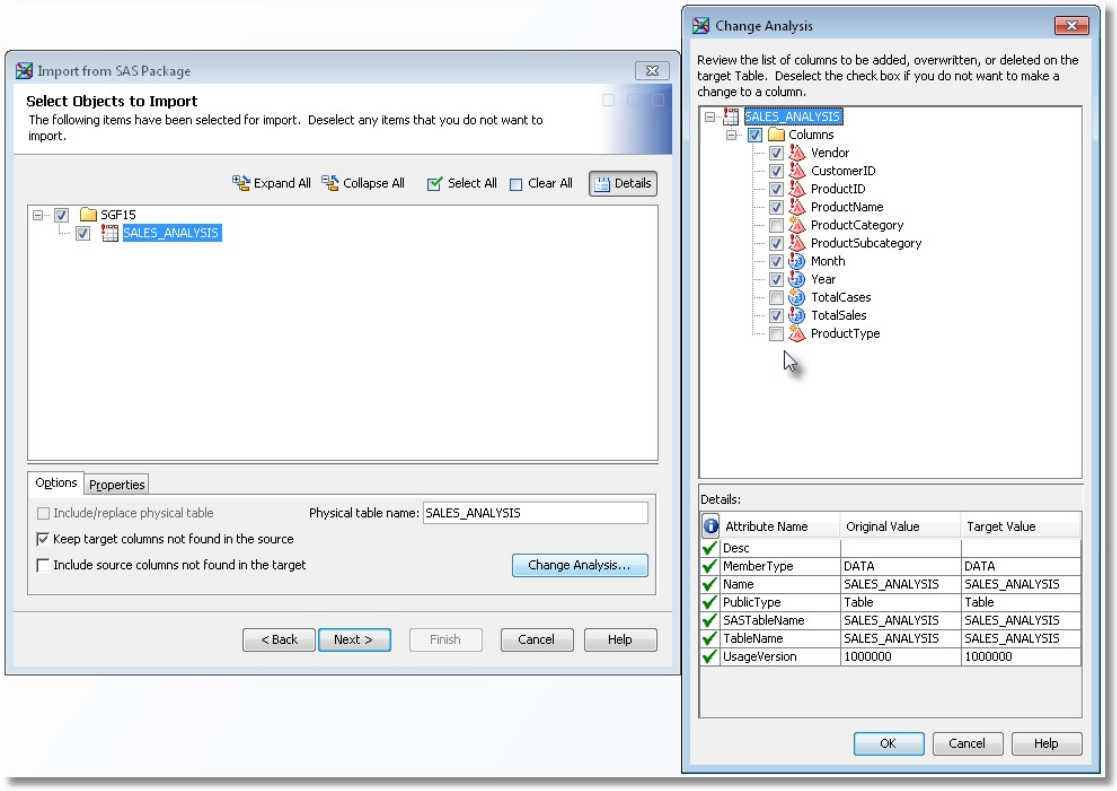


Figure 22

In the example shown in Figure 22 for the SALES_ANALYSIS table, by de-selecting the 'Include source columns not found in the target' check box the new metadata variables for ProductCategory, ProductType and TotalCases wouldn't be imported and over-write the existing table metadata.

CONCLUSION

In this paper we have seen how metadata and physical data can become out-of-sync. We have also been introduced to the MetaToPhysicalTableComparisonTool. This simple macro queries the metadata server and physical tables to extract basic table attributes. The attributes from both the metadata server and physical tables are then compared, and any anomalies noted.

We have also seen how the MetaToPhysicalTableComparisonTool can be used on entire libraries of data making tool a valuable addition to any SAS Administrators toolkit.

Finally we have seen how adding an additional argument into the SAS Data Integration initiation file can provide addition options when importing metadata between SAS Environments.

REFERENCES

[1] Muriel, Elena. "Exploring the Metadata Family Tree" Proceedings of the SAS® Global Forum 2009 Conference

[2] SAS 9.3 Open Metadata Interface Reference & Usage

[3] SAS 9.3 Language Interface to Metadata.

ACKNOWLEDGMENTS

CONTACT INFORMATION

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

David Moors
Whitehound Limited
21 Lyons Lane
Appleton, Cheshire, WA4 5JG
England
whitehound.limited@gmail.com

APPENDIX

```
/* assign METADATA library */
libname SGF15 meta library="SGF15";

/* assign PHYSICAL library */
libname sgf15p 'S:\SGF_Paper_Files\2015\data';


options nomprint nosymbolgen nomlogic;
%macro meta_compare(MtableName=, plib=, PtableName=);

   /* debugging - remove when ready */
   %put Physical Table Name: >>> &PtableName >>>;
   %put Metadata Table Name: >>> &MtableName >>>;

   /* check Physical table name is empty. Assign metadata name to physical name macro variable
   */
   %if &PtableName = %then %do;
      %let PtableName = &MtableName;
      %put Physical Table Name is now called: >>> &PtableName >>>;
   %end;


   /*-----------------------------------------------------*/
   /* obtain metadata view of the columns in the dataset */
   /*-----------------------------------------------------*/
   %let metaDSN=work.metadata_columns;

   data &metaDSN(keep=column_name format_type column_length);
       length column_name $32.
              column_length $6.
              format_type $200.
              columnURI tableURI metatype $256.
              ;

       /* The CALL MISSING statement initializes the output variables to missing values. */
       call missing (of _all_);

       rc=metadata_resolve("omsobj:PhysicalTable?@Name='&MtableName'",metatype,tableURI);

       /* The METADATA_GETNASN function gets objects associated via the table URI.
       numcols=metadata_getnasn(tableURI,"Columns",1,columnURI);
```

```
        /* The DO statement specifies a group of statements to be executed as a unit.
           The METADATA_GETATTR function gets the values of the Column Name, Column Length and
           SAS Format attributes. */
        do colnum=1 to numCols;
           numcols=metadata_getnasn(tableURI,"Columns",colnum,columnURI);
           rc=metadata_getattr(columnURI,"SASColumnName",column_name);
           rc=metadata_getattr(columnURI,"SASColumnLength",column_length);
           rc=metadata_getattr(columnURI,"SASFormat",format_type);
           output;
        end;
    run;


    /* convert column names and formats to upcase */
    data &metaDSN;
       set &metaDSN;

       column_name=upcase(column_name);
       format_type=upcase(format_type);
    run;

    /* sort variables into alphabetical order */
    proc sort data=&metaDSN;
       by column_name;
    run;


    /*----------------------------------------*/
    /* export physical columns to a dataset */
    /*----------------------------------------*/
    %let physicalDSN=work.physical_columns;

    proc contents data = &plib..&PtableName noprint
                  out = &physicalDSN(rename=(name=column_name)
                                     keep=name length format formatl formatd
                                     );
    run;

    /* convert column names to uppercase */
    data &physicalDSN(keep= column_name column_length format_type);
       set &physicalDSN;

       column_name=upcase(column_name);
       column_length=put(strip(length),$6.);

       if formatl NE 0 and formatd > 0 then format_type=cats(format,formatl,'.',formatd);
       else if formatl NE 0 then format_type=cats(format,formatl,'.');
       else if formatl = 0 and formatd = 0 and format NE '' then format_type=cats(format,'.');
    run;

    /* sort variables into alphabetical order */
    proc sort data=&physicalDSN;
       by column_name;
    run;

    /*-----------------------------------------------*/
    /* compare metadata columns to physical columns */
    /*-----------------------------------------------*/
    proc compare base=&physicalDSN
                 compare=&metaDSN
                 ;
    run;
%mend meta_compare;
%meta_compare(MtableName=sales_analysis, plib=sgf15p, PtableName=)
```