Paper 138-2008

Metadata for Everyone

A Simple, Low-Cost Methodology

Timothy D. Brown, Altoona, IA

ABSTRACT

In the context of Base SAS® programming, this paper uses "hardcoded" values as an introduction to "metadata" and the reasons for using it. It then describes a low cost and simple methodology for maintaining any kind of metadata.

INTRODUCTION

This discussion will take an indirect approach to defining metadata. It'll describe the metadata which might be included, or hard-coded, in a Base SAS program and propose alternatives to storing and using the metadata.

Outside of programs "data" and "code" are distinct. However within programs, the distinction gets blurred when data values, called "hardcoded" data, are included within the code.

Hardcoded values include, but are not limited to:

- Text constants, literals
  - Names of companies, people, organizations and places
  - Directory paths and file names
  - Parameters on SAS procedures such as WHERE, KEEP, DROP, RENAME, VARS, BY etc
- Numeric constants including dates*
  - Statistical constants
  - Period begin and end dates
- Mixed text and numeric values
  - Expressions in IF and WHERE clauses
  - What-if scenarios

(* excluding dates which are derived logically using a SAS functions such as TODAY(), DATETIME(), INTNX and NXTPD)

In addition, many small conversion, cross-reference and look-up tables, which might be hardcoded as SAS formats or read into a program from many different sources, work well as metadata and fit into this framework.

Obviously, some hardcoded values might never change in the life of a program. So it might be prudent to leave some hardcoded values in the code. However, if there's any chance of a hardcoded value changing during the life of the program, then it might be better to treat it as metadata and take it out of the program.

The good news is that it is quite straightforward to remove hardcoded values from a SAS program and to still have the values available to the program logic just as conveniently as when the data was hardcoded. And this is done by treating the hardcoded values as metadata.

WHY USE METADATA?

Nearly every SAS User feels the dual pressures of "production" and "quality" because management and customers want results faster and better. Metadata helps to relieve these pressures because it is faster to develop and maintain metadata, and with a higher degree of quality, than it is to develop and maintain source code containing hardcoded values.

Since data by its very nature is more dynamic than source code, programs with hardcoded values will change more often to alter the hardcoded data values than to change the logic of the program. By removing the hardcoded data and treating it as metadata, the program code does not change when metadata changes, and so the quality of the program is the same before and after a metadata change.

Metadata changes can be treated differently than source code changes. This is detailed in the Change Management section below. But the key is that different testing approaches can be used with metadata which are faster than the

very stringent and time consuming testing which is generally required for changes to source code. As a result, changes can be implemented faster.

A hot topic in business today is “a single source of the truth”. Metadata supports this concept by having just one place where the one value for a parameter or constant is stored and managed.

Some of the techniques described below, particularly the ones which use Microsoft® Excel, help to improve communications between developers and business users. Because many users are very comfortable with Excel, they comprehend the metadata stored in Excel well and potentially, they can submit changes, especially metadata changes, more directly using Excel.

On a similar note, it is easy to develop documentation from metadata because it is stored in one place rather than scattered throughout one or more programs.

And the last but surely not the least reason for using metadata is that it is becoming an increasingly integral part of data processing and the sooner you start to understand it the better off you will be.

CAUTION

The reader must be cautious not to be over zealous with metadata. Many processes will function poorly when metadata is applied. Carefully evaluate each application of metadata to assure that it adds value to your process.

PROGRAMMING CHANGES

SAS provides the macro language, macro variables, formats and hashing to support using metadata instead of hardcoded values in the program. The use of these tools is beyond the scope of this paper. Please see support.sas.com or other SAS resources for more information on these topics.

METHODOLOGY

This approach will provide:

- A single source for metadata for an entire SAS session
- A simple user interface for maintenance

In its simplest implementation, the focal point of this low cost and simple methodology is **one file**. It can be a SAS dataset, text file, an Excel file or a SAS program. This discussion will continue with a focus on a metadata SAS program so the various kinds of metadata can be clearly depicted. The reader should remember that other options are available which might work better in their own situation.

One drawback to the SAS program approach is that the developer may be forced to apply extensive testing to implement changes. It might be best to keep the metadata in a SAS dataset, text or Excel file to which other, faster, testing or quality control procedures can be applied. A program would still be required to load the metadata into the SAS environment, but it too would not need to change every time a value in the metadata changed.

Nonetheless, continuing with the SAS program approach, while the general objective is to remove all hardcoding from the programs, placing the metadata in a single SAS program is a nice compromise. It removes hardcoded values from every other program and places it in one central, easy to maintain program.

Metadata must be accessible from anywhere during a SAS session. The following series of sample Data Steps demonstrate how different kinds of metadata can be setup in a “Metadata Load” program. Whenever the whole Metadata Load program executes, the metadata detail is loaded into the SAS environment where it will be available to any other SAS program which runs in the same SAS session. So it might be useful to include the Metadata Load program in the autoexec.sas program. (See %INCLUDE in the SAS documentation.)

Name-Value Pairs as Macro Variables:

```
data mvars;
   infile datalines dsd missover;
   attrib name label="Name" format=$50.;
   attrib value label="Value" format=$255.;
   input name $ value $;
   datalines;
mv_backup,backup  (B)
mv_current_history_yr,2006  (C)
mv_dirname,"development_system_%substr(&mv_mthp1_yyyymmdd,1,4)_%substr(&mv_mthp1_yyyymmdd,5,2)"
(A)
mv_err1,ERR  (B)
```

```
mv_err2,OR:
mv_loandata,loandata  (B)
mv_last_history_yr,2005
mv_metadata_file,&mv_pcms_dir\metadata\pcms_metadata.xls
mv_mode_cd, (N)
mv_mode_term,prod
mv_ms_access_dbms,access2000
mv_ms_access_exe_path,c:\program files\microsoft office\office10\msaccess.exe (D)
mv_ms_excel_dbms,excel2000
mv_output_attributes,&&rootdir/programs/output_attributes.sas  (E)
mv_pcms_dir,&mv_q_drive\pcm_prodops\pcms\&mv_mode_term  (F)
mv_gid_gl_joe_dir,&mv_q_drive\portfolio_risk\reference\gl joe  (F)
mv_gid_nonrel_rate,5.375  (C)
mv_gid_rel_rate,5.125  (C)
mv_gid_reporting_dir,&mv_pcms_dir\output\gid\reporting  (F)
mv_prodrm_data_start_sas,'01jan2000'd  (G)
mv_prodrm_where_app_loans,'200001' <= date_vintage_year_month_app <= &yrmn  (H)
mv_prodrm_where_subprime,"wfmr_type in ('Subprime', 'SECOND', 'ENORA', 'NORA')"  (I)
mv_projects_archive_dir,/projects/archive/
mv_wh,loan_id lt 'zzzzzzzzzzz' (H)
;
run;
data _null_;
   set mvars;
   call symput(cats(name),cats(value));
run;
```

(Note: "cats" is a new function as of SAS 9.1 which simplifies the "trim(left())" functions.)

The details above which **follow the DATALINES statement** and **end at the semi-colon before RUN** are ***Name-Value pairs***. The Name occurs first on each line followed by a comma and the Value. The in-stream data above is formatted in the same Comma Separated Values (CSV) format as would be found in a .csv file.

(A text file for metadata should also be formatted as a .csv file.)

Each line in this data, except the line labeled **(A)** is a single record in the output file. Line **(A)** shows that it is also acceptable to have a record extend beyond one physical line. The length of the Name and Value fields is set by the ATTRIB statements at the top of the Data Step. Of course FORMAT and/or LENGTH statements could be used instead of the ATTRIB statement. It's very important to establish the length of the fields at the beginning of the Data Step; otherwise, the length of the fields in the first record will set the lengths for the respective fields for all the other records.

The INPUT statement establishes the fields which will be read into the step and whether they are character or numeric fields. In the step above, two character fields, NAME and VALUE will be read. The MISSOVER option will set the value of the missing field(s) to missing (blank for character fields and .(missing) for numeric fields).

It's good practice to code the MISSOVER option. It is also good practice to explicitly code a value for each field on each data record. A comma is not placed after the last field in each line. As a result each data record will have *((the number of fields) minus 1)* commas. No semi-colons are used; this is data not SAS code. If the last field in a line is null, nothing is coded. If any other field is null, then just one comma (for each null field), should be coded (see line **(N)** above). This approach makes it easier for subsequent programmers to read the data.

In this example, all the Names were arbitrarily prefixed with "MV_" to prevent a conflict between the names of these "system" macro variables and the names of macro variables which might be used in another SAS program in the same SAS session.

The second Data Step reads the data from the first step and converts the Name-Value pairs into macro variables where the Name becomes the macro variable name and the Value becomes the value of the macro variable. This Data Step will also set the scope on all the macro variables to **Global** which in turn will make these variables available to every subsequent program in the SAS session.

Notice the numerous types of metadata which can be stored in this manner:
(B) – literal or text constants
(C) – numeric constants
(D) – Microsoft Windows directory names

(E) – UNIX path names
(F) – text strings containing macro variables
(G) – date constants
(H) – the value of a WHERE clause (or any other piece of SAS syntax)
(I) – text strings containing commas and quotes

And note that in (N), MV_MODE_CODE has a null value

Dates as Macro Variables:

Static or slow changing dates can be stored as shown in (G) above. And dates which are driven by logic and the today() or datetime() functions are fine in the program. However, if there's any possibility that the value of "today" will change, then it would be a good practice to build "today" override logic into program and the override value could be provided via the metadata. For example, perhaps a storm shuts down operations and after the storm you have to run programs as-if they were an earlier date.

It could be as simple as setting up a macro variable, let's say 'MV_RUNDATE_ OVERRIDE, in the Data Step above. And in the logic, you could code:

```
if "&MV_ RUNDATE_OVERRIDE"="" then td=today();
else td="&MV_ RUNDATE_OVERRIDE";
```

This technique prevents the need for a change to the program code to rerun the program or system on other dates.

Name-Value Sets as SAS Formats:

Sometimes the metadata is structured in groups or sets which can be used as SAS Character and Numeric Formats. Such format could be stored in the Metadata Load program (as a central location) instead of having them hardcoded through the program.

```
/*************************************************************************/
/*                          Character Formats                            */
/*************************************************************************/
data cformats;
   infile datalines dsd missover;
   attrib fmtname label="Format Name" format=$8.;
   attrib start label="Start" format=$20.;
   attrib end label="End" format=$20.;
   attrib label label="Label" format=$35.;
   attrib hlo label="HLO" format=$8.;
   input fmtname $ start $ end $ label $ hlo $;
   /* VERY IMPORTANT!!                                                   */
   /* Missing values lines MUST have a letter "O"(other) in the HLO field!!!  */
   datalines;
$jumbo,Y,Y,Y,
$jumbo,,,N,O
$stated,Stated,Stated,Y,
$stated,,,N,O
$conflmt,N11990,N11990,281175,
$conflmt,N11991,N11991,286875,
$conflmt,N11992,N11992,303450,
$conflmt,N11993,N11993,304725,
$conflmt,,,99999999,O
;
run;

/*************************************************************************/
/*                            Numeric Formats                            */
/*************************************************************************/
data nformats;
   infile datalines dsd missover;
   attrib fmtname label="Format Name" format=$8.;
   attrib start label="Start" format=best.;
   attrib end label="End" format=best.;
   attrib label label="Label" format=$35.;
   attrib hlo label="HLO" format=$8.;
   input fmtname $ start $ end $ label $ hlo $;
   datalines;
def,0,0,No Default,
def,1,1,Default,
```

```
dtlnsrc,1,1,1. date_funded,
dtlnsrc,2,2,2. date_closing,
dtlnsrc,3,3,3. date_note,
dtlnsrc,4,4,4. date_first_pymt_due_note,
dtlnsrc,,,missing,O
;
run;

/*****************************************************************************/
/*        Load Character and Numeric Formats into SAS Environment            */
/*****************************************************************************/
proc format library=formats cntlin=cformats; run; quit;
proc format library=formats cntlin=nformats; run; quit;
```

Attributes as a SAS Data Set – 1:

The program from which this SAS code was taken was very flexible. It could read and write numerous files. And each file could be processed with a variety of functionality. This next step shows a list of files; the LIBNAME; and whether the file was to be used as input(i), output(o) or both (Work W). The other fields in each record are parameters, or metadata, that control which functionality was applied to each file.

```
/*****************************************************************************/
/*                              File Control                                 */
/*****************************************************************************/
data file_control;
   infile datalines dsd;
   attrib library label="Library" format=$20.;
   attrib file label="File" format=$40.;
   attrib input_output label="Input_Output" format=$1.;
   attrib rec_cnts label="Capture Rec Cnts" format=$1.;
   attrib keep label="Use Keep Clause" format=$1.;
   attrib rename label="Use Rename Clause" format=$1.;
   attrib missing_loan_ids label="Bypass Missing Loan_ID Recs" format=$1.;

   input library $             /* Required field */
         file $                /* Required field */
         input_output $        /* Required field */
         rec_cnts $
         keep $
         rename $
         missing_loan_ids $;
   datalines;
custdata,borrower,I,Y,Y,,Y
custdata,investor_xref,I,Y,Y,,
custdata,loan,I,Y,Y,,Y
custdata,serviced_loans,I,Y,Y,,Y
custdata,servicing_current,I,Y,Y,,Y
loandata,delinquency_out,O,,Y,Y,
loandata,investor_xref_out,O,,,,
loandata,more_loan_detail_out,I,,Y,,Y
loandata,more_loan_detail_out,O,,Y,,
tempdat,loans_app2,W,,,,
tempdat,temp_score_current,W,,,Y,
tempdat,temp_serviced_loans,W,,,,
tempdat,temp_servicing_current,W,,,Y,
;
run;
```

Again, the data in this step is formatted as a Comma Separated Values (CSV) file would be, but this is in-stream data.

This is a good time to mention that it is easier to work with CSV data by placing the data, without any SAS code, in a .csv file and opening the file using Microsoft Excel. If the data changes, be sure to save the new file as a .csv file. This is one reason why it might be advantageous to store the in-stream data as separate .csv files.

But also note that Excel sometimes changes the data in ways which do not work well in SAS. For example, if you store SAS formats such as “8.”, Excel might make it “8.00”. If Excel is used to maintain the metadata then all the metadata should be explicit character values – even numerics.

After the updates are saved, open the new .csv file using a simple text editor such as Notepad, copy the data and paste it back into the SAS program.

Attributes as a SAS Data Set – 2:

This code is from an extract-transform-load (ETL) application in which the input and output fields frequently changed. As a result, **all** the fields from **all** the input files, and **all** the respective field attributes, were included as the in-stream data in the original data step.

```
/*************************************************************************/
/*                                I/O In                                 */
/*************************************************************************/
data i_o_in;
   infile datalines dsd missover;
   attrib keep label="Keep" format=4.;
   attrib library label="Library" format=$20.;
   attrib file label="File" format=$25.;
   attrib variable label="Variable" format=$32.;
   attrib length label="Length" format=4.;
   attrib format label="Format" format=$15.;
   attrib label label="Label" format=$50.;
   attrib varnum label="Varnum" format=4.;
   attrib type label="Type" format=$4.;
   attrib sortkey label="Sortkey" format=4.;
   attrib rename label="Rename" format=$32.;
   input keep library $ file $ variable $ length format $ label $
         varnum type $ sortkey rename $;
   datalines;
1,custdata,borrower,loan_id,12,$12.,Loan ID,4,char,100,
1,custdata,borrower,fico_b,4,,Fico Score - B1,42,num,300,
0,custdata,borrower,alltel_client,3,$3.,Alltel Client,3,char,400,
0,custdata,borrower,age_b,4,,,5,num,1000,
0,custdata,borrower,age_oldest_trade_b1,4,,Age of Oldest Trade,9,num,1000,
0,custdata,loan,date_credit_received,4,yymmdd10.,CREDIT RECEIVED DATE,61,num,1000,
0,custdata,loan,date_rate_lock,4,yymmdd10.,RATE LOCK DATE,71,num,1000,
0,custdata,loan,date_undwr_submission,4,yymmdd10.,UNDERWRITING SUBMITTED DATE,72,num,1000,
0,custdata,loan,date_vintage_year_app,4,$4.,,74,char,1000,
0,custdata,loan,decision_site_code,6,,DECISION_SITE_CODE,79,num,1000,
0,custdata,loan,decisioners_uw_auth_lvl_code,2,$2.,DECISIONERS_UW_AUTHORITY_LEVEL_CD,77,char,1000
,
0,custdata,loan,division_number,3,,DIVISION_NUMBER,83,num,1000,
0,custdata,loan,documentation_type_raw,1,$1.,DOCUMENTATION TYPE - RAW,85,char,1000,
1,custdata,loan,repurchase_flag,1,$1.,Repurchase Flag,154,char,1000,
1,custdata,loan,reserves,6,,Reserves,155,num,1000,
1,custdata,loan,satellite_region_code,4,$4.,Satellite Region Code,157,char,1000,
1,gid,prm_current,onegl_entity,3,$3.,OneGL_Entity,1,char,1000,
1,gid,prm_current,pcrm_reporting_segment,10,$10.,PCRM Reporting Segment,2,char,1000,
1,gid,prm_current,reo,1,$1.,Reo,12,char,1000,
1,gid,prm_current,reo_investor,3,$3.,Reo Investor,13,char,1000,
1,gid,prm_current,service_monitor,1,$1.,Service Monitor,14,char,1000,
1,gid,prm_current,serviced_others,1,$1.,Serviced Others,15,char,1000,
;
run;
```

The KEEP field was used to include(1) or exclude(0) a particular field from the file.

The LIBRARY, FILE, and VARIABLE fields defined each field of data.

The LENGTH, FORMAT, LABEL, VARNUM and TYPE field provided the means to redefine the input data in any manner which was needed for the application.

The SORTKEY field provided a way to organize the variables in the files. In this case, key fields were assigned low number so they would appear at the beginning of each file, and all the other fields were assigned the same number and then sorted alphabetically.

And the RENAME field provided the means to dynamically rename files.

Since the metadata contained a similar step for all the output files, the System Administrator had substantial – and easy – control over which files would be would be read in and out, as well as which fields on every field where to be included, as well as the exact attributes, even the name(rename) of every variable.

All this control and flexibility were accomplished without changing the code in any program in the application except the metadata program.

Conversion Table as a SAS Data Set:

This step contains a conversion, or cross-reference, table.

```
/*************************************************************************/
/*                          Conforming Values                            */
/*************************************************************************/
data comforming_values;
   infile datalines dsd;
   attrib in_continental_usa label="Lives In Continental USA" format=$1.;
   attrib living_units_number label="Living Units Number" format=$1.;
   attrib year label="Year" format=$4.;
   attrib conv_loan_limit label="Conventional Loan Limit" format=$6.;
   input in_continental_usa $ living_units_number $ year $ conv_loan_limit $;
   datalines;
N,1,1990,181175
N,2,1990,219625
N,3,1990,244625
N,4,1990,340225
Y,1,2005,359650
Y,2,2005,560400
Y,3,2005,656500
Y,4,2005,791600
;
run;
```

The first 3 columns are the key fields which are used to identify the value in the fourth column.

Obviously, many SAS applications use conversion tables which are substantially larger. It would be up to the developer to decide when a conversion table would be better as a distinct SAS table or as an in-stream table as shown above. However, for all those little annoying tables which are scattered throughout your department, this approach provides a means to store and manage those tables in a central location.

ALTERNATIVES

The alternatives to maintaining metadata in a SAS program are:

1. SAS datasets but the updating would generally be cumbersome
2. files in Oracle, DB2, etc but the metadata would not be centralized
3. one or more Excel files. While Microsoft Excel is more user friendly than many alternatives, it can't run in some environments.
4. individual text files containing data in the CSV format and read them into the SAS program using an **INPUT** or **%INCLUDE** statement

In addition, any combination of these alternatives can be used.

Metadata storage and maintenance can be as complex or as straight forward as the work requires. But as the physical design becomes increasing decentralized, the risk to control and quality increase as well.

DEVELOPMENT

It might be advantageous to have separate Load Metadata programs for Production and for each developer. This would enable developers to change the values in their programs as needed for each project and for staging before they are implemented in Production.

CHANGE MANAGEMENT

Procedures for migrating changed metadata to testing environments and ultimately into Production would need to be worked out wherever these concepts are applied.

Metadata change management frequently requires a new role in the organization - a "gatekeeper". This role would be very similar to that of the Security Administrator who receives requests to build new access groups, add people to

groups, and authorize groups to access new resources. A Metadata Administrator would need to receive management authorization before making a change, to understand the impact of change and to implement a change very carefully to avoid adverse impacts.

Since this approach still stores the metadata within a SAS program, standard change management may still be required.

Additional auditing procedures may also be required to monitor, the sometimes very small, changes which can be made in metadata. Some tools are available on the Internet for comparing an original and a modified text file (SAS program). Such a program could be run and all the changes could be validated against the authorized changes, before a metadata program is moved into a production environment.

For quality control and auditing purposes it might also be beneficial to write a program to validate all the metadata every time it's loaded into the SAS environment.

Another auditing requirement might be to demonstrate the actual current value of metadata as a program starts to execute. The requirement might be to print all the metadata in the log every run. The **%PUT _ALL_** SAS command displays all the macro variables in the log and PROC PRINT can be used to dump small tables to the log.

CONCLUSION

Metadata can be defined as narrowly or as widely as the situation requires. Do not let arguments about the definition of metadata prevent you from improving your process to gain all the potential value from metadata.

This paper does not present any rules. If a portion of the metadata is better stored in a dataset, or even in a file in another system such as a table in an Oracle database, then so be it.

This approach works best for a single user or a small organization which needs to manage bits and pieces of disparate details in order for their systems or analyses to execute effectively.

Metadata in a SAS program is generally hardcoded data. There is true value in removing hardcoded values from programs.

Value can also be gained by centralizing the storage and maintenance of metadata.

The techniques presented in this paper are inexpensive and easy to use. And they will help you to manage your metadata today.

ACKNOWLEDGEMENTS

I would like to thank Scott Burkland for his contributions to this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author at:
Tim Brown
(302) 690-7650
timbrown74@yahoo.com