



FMD K&L

*Programming Solutions when Developing a
Database Compare Macro*

(CT13 - PhUSE 2017)

Michael S Rimler, FMD K&L Inc

Overview

- Utility macro which compares SAS datasets from different versions of the same database
- Designed to
 - Require a relatively simple call (easy to use)
 - Robust to incorrectly specified parameter values (difficult to break)
 - Generate meaningful reports of discrepancies (easy to adopt)

Motivation

- Programming groups receive regular data transfers when much of the programming is already in place
- Role of macro:
 - Compare inputs of programming system
 - Check stability of database design prior to execution of system
 - Identify changes in input data to predict expected changes in mapped database or TFLs
 - Compare outputs of programming system
 - When input database changes
 - When programming infrastructure changes

Macro Parameters

Parameter	Description	Default Value
BASE_PATH	Global path of original SAS database	
COMP_PATH	Global path of updated SAS database	
DSET_LIST	List of datasets to compare (= _all_ will compare all datasets)	_all_
EXCL_DSET	List of datasets to exclude	
SORTVARS2INCL	Variables prioritized in sort key search algorithm	
SORTVARS2EXCL	Variables deprioritized in sort key search algorithm	
PERMDSETYN	If = Y, output permanent datasets	N
PERM_PATH	Global path of permanent datasets to be output	
MAJUPDYN	If = Y, suppresses record level comparisons (Reports 6-8)	

Sample Call

```
%db_compare(base_path      = %str(Z:\user\mr\basedata\)  
            ,comp_path     = %str(Z:\user\mr\compdata\)  
            ,dset_list      = _all_  
            ,excl_dset      = LB PC  
            ,sortvars2incl  = USUBJID *CAT *TESTCD *DTC  
            ,sortvars2excl  = *SEQ  
            ,permdsetyn     = Y  
            ,perm_path      = %str(Z:\user\mr\outdata\)  
            ,majupdyn       = Y  
            );
```

- All datasets, except LB and PC, will be compared
- The sort key algorithm will prioritize variables in **sortvars2incl**
- The sort key algorithm will deprioritize variables in **sortvars2excl**
- Permanent datasets of reports will be created
- Record level reports will be suppressed

Challenges to Development

- Macro parameter validation
- Dataset specification
- Robust dataset comparisons – deriving a unique sort key
- User control of sort key search algorithm
- Addressing domains with fully replicated records
- Designing output generated by the macro (discrepancy reports)

Macro Parameter Validation

- Verify required parameters are provided

```
%if &base_path eq %then %do;  
    %put %str(E)RROR: DB_COMPARE: The parameter BASE_PATH is required.;  
    %goto EndMac;  
%end;
```

- Verify paths to databases exist on the system

```
%local rc fileref;  
%let rc=%sysfunc(filename(fileref,&base_path));  
%if %sysfunc(fexist(&fileref)) %then %do;  
    libname BaseData "&base_path.";  
%end;  
%else %do;  
    %put %str(E)RROR: DB_COMPARE (base_path): The directory "&base_path" does not exist.;  
    %put %str(E)RROR: DB_COMPARE (base_path): Processing will be stopped.;  
    %goto EndMac;  
%end;
```

Macro Parameter Validation (continued)

- Remove case sensitivity

- Example 1

```
if %upcase(&permdsetyn) eq Y or %upcase(&permdsetyn) eq YES %then %do;
```

- Example 2

```
%let l_dset_list=%lowcase(&dset_list.);
```

- The developer determines the syntactic requirements of the utility macro and implements validation code for enforcement
- Messages to the user (via the log) assist the user in resolving issues

Dataset Specification

- Macro must be informed on which datasets to compare
- Two parameters perform this function
 - **dset_list**
 - **excl_dset**
- **DSET_LIST**
 - Explicit space-delimited list of datasets to include in comparison
 - Macro allows for **_all_**, which will construct a list of all existing datasets using PROC CONTENTS from each database
- **EXCL_LIST**
 - Explicit space-delimited list of datasets to exclude from comparison
 - Used in conjunction with **dset_list = _all_**

Dataset Specification (continued)

- Pseudocode

```
%if &l_dset_list eq %str(_all_) or &l_dset_list eq %then %do;  
  /* Insert code for PROC CONTENTS on both libraries */  
  /* Set together and retain all unique MEMNAME values */  
  /* Exclude values specified in &excl_dset */  
  /* Populate resulting list in &full_dset_list */  
%end;  
%else %do;  
  %let full_dset_list=&dset_list;  
%end;
```

- Sample call

```
%db_compare(base_path      = %str(Z:\user\mr\basedata\  
              ,comp_path    = %str(Z:\user\mr\compdata\  
              ,dset_list     = _all_  
              ,excl_dset     = LB PC  
              );
```

Dataset Specification (continued)

- The user-provided list of datasets are processed by merging with PROC CONTENTS result of each database
- A specified dataset that is non-existent will not result in execution error
- For example, **dset_list = AF LB DM ADSL** on an SDTM database will only process **LB** and **DM**
- Adds robustness to the syntactic requirements of the macro
- Reviewing the log and output will assist identifying user error

Deriving a Unique Sort Key

- Direct comparison of source datasets can be meaningless if not sorted similarly
- Macro derives a sort key endogenously from data through a search algorithm
- Variables common to both source datasets are grouped into 4 tiers, establishing a priority level
- Limited influence on search algorithm is provided to user (**sortvars2incl** and **sortvars2excl**)

Assignment of Tiers



Tier 1: Variables specified in sortvars2incl



1. Tier 4: Variables specified in sortvars2excl



Tier 2: Remaining variables included in SORTEDBY metadata from PROC CONTENTS, in sort order



Tier 3: Remaining variables in VARNUM order (left to right)

Deriving a Unique Sort Key (continued)

- Initial Conditions
 - **StopFlag = 0**
 - **NumVars = 1**
 - **VarList = null**
 - **PrevNumObs = $N_{\text{BASE}} + N_{\text{COMP}} + 1$**
- For each dataset, execute **DO UNTIL** loop (until **StopFlag = 1**)
- In each iteration,
 - Set test sort key to VarList + Next Variable
 - Execute PROC SORT NODUPKEY using test sort key on COMP data
 - Retain number of duplicate observations deleted

Deriving a Unique Sort Key (continued)

- Stopping conditions

- Number of duplicates removed = 0

```
*** If number of dups = 0, stop ***;  
%if %eval(&NumObs.)=0 %then %do;  
    %let StopFlag=1;  
%end;
```

- Tested last common variable in dataset (and duplicates remain)

```
*** If tested last variable, stop ***;  
%if %eval(&NumVars.)>%eval(&MaxNumVars.) %then %do;  
    %let StopFlag=1;  
%end;
```

Deriving a Unique Sort Key (continued)

- If stopping condition is not met, but number of duplicates decreases, retain variable to sort key

```
*** If number of dups decreased, keep variable in list ***;  
%if %eval(&NumObs.) < %eval(&Prev_NumObs.) %then %do;  
    %let varlist = &varlist. &AddVar.;  
    %let Prev_NumObs = &NumObs.;  
%end;
```

- If stopping condition is met, test current sort key on BASE data
 - If number of duplicates removed equals zero, stop search
 - If number of duplicates is greater than zero, restart search on BASE data
 - Initialize sort key to current **VarList**
 - Skip any variables already in **VarList** (for efficiency)
 - Use the same stopping conditions for search on BASE data

User Control Over Sort Key

- Limited influence on search algorithm is provided to user (**sortvars2incl** and **sortvars2excl**)
- **SORTVARS2INCL**: Space-delimited list of variables to prioritize (grouped in Tier 1)
- **SORTVARS2EXCL**: Space-delimited list of variables to deprioritize (grouped in Tier 4)
- Similar processing and syntactic requirements as **dset_list** and **excl_dset**
- Added Feature: Wildcard character ‘*’

User Control Over Sort Key (continued)

- Pseudocode

```
data outdata;  
  set indata;  
  %do ii = 1 %to %eval(&neqi.);  
    %if &ii > 1 %then else;  
      if upcase(NAME) = "&&veqi&ii." then TIER=1;  
    %end;  
    %do ii = 1 %to %eval(&ncoi.);  
      else if find(upcase(NAME), "&&vcti&ii.") > 0 then TIER=1;  
    %end;  
    %do ii = 1 %to %eval(&neqe.);  
      else if upcase(NAME) = "&&veqe&ii." then TIER=4;  
    %end;  
    %do ii = 1 %to %eval(&ncoe.);  
      else if find(upcase(NAME), "&&vcte&ii.") > 0 then TIER=4;  
    %end;  
    else if B_SORTEDBY^=. | C_SORTEDBY^=. then TIER=2;  
    else TIER=3;  
  run;
```

Fully Replicated Records

- Difficult to compare if one dataset has records that are 100% replicated on all common variables
- Solution:
 - Remove records from further analysis of dataset
 - Report records directly to user
 - Annotate reports using reduced dataset that replicated records have been removed
- Requires a method for identifying replicated records: result of sort key algorithm

Reports to the User

- Three types of reports are generated
 - PROC PRINT reports to the output window or external file (.LST)
 - Temporary datasets supporting the PROC PRINT results
 - Permanent datasets identical to temporary datasets (**permdsetyn = Y** and **perm_path**)
- Three levels of reports are generated
 - Database level reports
 - Dataset level reports
 - Record level reports

Reports to the User (continued)

- **Database Level Reports:**
 - **Report 1:** Datasets with at least one difference identified
 - **Report 2:** Datasets which do not exist in one database or the other
 - **Report 3:** Datasets with different number of observations
- **Dataset Level Reports:**
 - **Report 4:** Variables which do not exist in one dataset or the other
 - **Report 5:** Metadata Differences
- **Record Level Reports**
 - **Report 6:** Records which do not exist in one dataset or the other
 - **Report 7:** Common records with value differences
 - **Report 8:** PROC COMPARE of datasets

Reports to the User

- **Additional Reports:**
 - **Datasets:** Complete datasets from each database, sorted by unique sort key, including duplicate records
 - **Replicated records:** Listing of fully replicate records
- Record Level and Additional Reports are suppressed if **majupdyn = Y**

End of Program

Thank you

Michael S Rimler, FMD K&L Inc

michael.rimler@klserv.com

michael.rimler@gmail.com