Macros with Global Vision: Using The SAS® Dictionary Tables to Create Tools
Michael E. Tomb and James R. Carter
Informatica ECS, Inc., Rochester, NY

ABSTRACT
The SAS Dictionary Tables are views of continually updated data describing the current SAS session. In this poster we provide examples of using this valuable resource of metadata and putting it to work. Each of the three macros presented here automates a solution that involves multiple SAS datasets and libraries. The first illustrates how to instantly compare all like-named datasets within two libraries based on date and a set of ID variables. The second is a simple character "word" search and replace routine that crawls across datasets and libraries. The last macro system reports tabular observation counts of all levels of ID variables after sweeping through every dataset and library that is desired. It can be used, for example, to streamline the process of tracking missing CRF pages within Clinical Trial data.

INTRODUCTION
In our opinion, the best macro tools are like good detectives. They inspect the surrounding environment to find out as much about the problem at hand as possible. And when doing their job they require a minimal amount of supervision and instruction. So, whenever possible, our tools utilize global vision within the macro code. We use "global vision" to refer to the technique of examining SAS data libraries and catalogs, and then writing code that automatically incorporates the names of entries, libraries, datasets, formats, variables and other components of the SAS data dictionary. Our macro vision is enabled through views of a session's metadata that are always provided by the SAS system. These views are known as the SAS Dictionary tables. In addition, we often make use of the CALL EXECUTE data step technique to write dynamic SAS code. The reader will want to be familiar with both SAS features to gain maximum benefit from this poster.

We begin with a simple example and then expand the scope of complexity while demonstrating two very useful tools.

SOLUTION #1
COMPARING MULTIPLES PAIRS OF DATASETS

INSPIRATION
After making major enhancement to an existing SAS application, we often run both the new and old systems in parallel for a time. For a system that updates hundreds of datasets, we needed a bulk method of using PROC COMPARE to compare entire libraries of datasets. Because so many datasets were involved, we often wanted to limit compares to only those datasets that had changed within a certain testing period. So we wrote a macro that taps into the SAS data dictionary environment and sets up a series of comparisons based on the modification date of datasets.

EXAMPLE
For the example we used the macro as follows:

CALL COMPLIBS(LIBREF=MASTERLB,LIBCOMP=UPDATELB,AFTER=’08FEB01:00:00:00’dt);

See Table 1 for an example of output.

CALLING SYNTAX
COMPLIBS is called using the following arguments:
LIBREF= a base LIBREF containing datasets
LIBCOMP= a Libref to compare against the base library
ID= Optional – a set of ID variables to use in the comparison
AFTER= Optional – a SAS datetime constant representing the date of last modification of an individual set of either BASE or COMPARE datasets

COMMENTED SOURCE CODE FOR MACRO COMPLIBS

As PROC SQL can directly select columns from the SAS dictionary tables, a join is used to quickly produce the list of all liked named datasets within two specified libraries. The same system table that contains the list of all dataset names also records other useful information such as the date of last modification for every dataset and the current Libref.

/*
MACRO COMPLIBS(LIBREF= WORK,LIBCOMP= WORK ,
ID = _NONE_, AFTER = _NONE_,
COMPOPTS = LISTALL );
*/
PROC SQL; CREATE TABLE TASKLIST as
SELECT A.MEMNAME AS DATASET , A.MODATE AS AMODATE , B.MODATE AS BMODATE ,
A.MEMTYPE FROM
(SELECT * FROM DICTIONARY.TABLES
WHERE LIBNAME="%UPCASE(&LIBREF)"
) A,
(SELECT * FROM DICTIONARY.TABLES
WHERE LIBNAME="%UPCASE(&LIBCOMP)"
) B
WHERE A.MEMNAME = B.MEMNAME AND A.MEMTYPE = "DATA" AND B.MEMTYPE = "DATA" ;
/

While the data step below is used to optionally subset the list of comparisons according to the date of last modification, this action could have been combined with the previous WHERE clause. But a separate step makes the logic a little easier to follow...

/*
%IF NOT ( &AFTER = _NONE_ ) %THEN %DO;
DATA TASKLIST ; SET TASKLIST;
WHERE MAX(AMODATE,BMODATE) > &AFTER ;
RUN;
%END;
*/

See Table 1 for an example of this PROC PRINT that lists all datasets for which a comparison will be generated.

/*
PROC PRINT LABEL DATA=TASKLIST;
VAR DATASET AMODATE BMODATE;
LABEL DATASET = "Dataset"
AMODATE = "Last Date Modified in &LIBREF"
BMODATE = "Last Date Modified in &LIBCOMP";
TITLE "DATASETS TO BE COMPARED WITHIN LIBRARIES &LIBREF AND &LIBCOMP";
RUN;
*/

CALL EXECUTE enables a system independent method for writing dynamic SAS code. (It can also make for difficult debugging!) In this instance, the CALL EXECUTE routine is used to write out a complete PROC COMPARE step for each pair of datasets that are going to be compared. In later examples, the choice of single quotes within the calling argument becomes critical.

/*
DATA _NULL_; SET TASKLIST END=EOF;
CALL EXECUTE("PROC COMPARE DATA=LIBREF.",&COMPOPTS ;");
*/

CALLING SYNTAX
macro. See Table 2 for an example of what is written to the log by this
LIBTRAN(FROMSTR="VALJEAN", TOSTR="JAVERT");
For the example we used the macro as follows:
known dataset is searched for the target string! )
by default every character variable in every observation in every
specifying a list of character variables, datasets and/or libraries,
(Be careful when testing these macros! If you don't restrict it by
library names. [There is a reason that all the data step variables local to this
each character variable in every observation in every
known dataset is searched for the target string!]

IMPROVEMENTS AND SIMILAR SOLUTIONS
While the macro “as is” is perfectly suitable for doing whole scale
comparisons, there are a couple of things that could extend its
behavior. First, we might thoroughly parse the “AFTER” string for
date-like content, as most users may not be familiar with SAS
DATETIME constant syntax. Second, we might include an option
that scans the output of PROC CONTENTS for sort variables and
then customizes the ID statement in the compare accordingly and
automatically (!)

SOLUTION #2
REPLACING STRINGS EVERYWHERE AT ONCE
INSPIRATION
Many programmers are familiar with editors and other tools that
enable wide-ranging search and replace operations. Such tools
may scan multiple file paths, directories and folders according to
custom options set by the user. The combined system of three
macros described here gives the SAS programmer the similar
ability to find and replace strings in a specified list of character
variables within all SAS datasets known within a declared list of
library names.

(Be careful when testing these macros! If you don’t restrict it by
specifying a list of character variables, datasets and/or libraries,
by default every character variable in every observation in every
known dataset is searched for the target string!)

EXAMPLE
For the example we used the macro as follows:
LIBTRAN(FROMSTR="VALJEAN", TOSTR="JAVERT");
See Table 2 for an example of what is written to the log by this
macro.

CALLING SYNTAX
LIBTRAN is called using the following arguments:
LIB= a list of LIBREFS, default is to scan ALL libraries
DS= A list of datasets names, default is to use all
datasets within a libraries
VAR= a list of variables to do the search and replace
on. Default is to search all characters within the
set of LIBREFS and datasets being processed. If you do specify the list, then
every variable in the list must be in every
dataset to be processed.
FROMSTR= a Character Constant containing the target
string to find
TOSTR= a Character Constant containing the
replacement string
LOG= YES writes a list and summary of all changes to
the SAS log, NO does not

COMMENTED SOURCE CODE FOR LIBTRAN
/
Part 1 MACRO CHGSTR –
While this macro was designed to be called automatically by the
main routine in this system (See LIBTRAN below), it can also be
used separately. The macro replaces all occurrences of a string
that represents a word within a list of character variables in a data
set. The _VAR argument is an optional list of SAS Character
variables...as the default behavior is to sweep through all
character variables within the dataset using the TRNWRD
function. If _LOG_ON is set to YES then a record of changes
and counts are written to the SAS LOG
*/
%MACRO CHGSTR(_DS=, _VAR=_CHARACTER_,
_FROMSTR=, _TOSTR= , _LOG_ON=NO );
/*
DATA &_DS ;
MODIFY &_DS END=DONE;
RETAIN _MODCNT _LINCNT 0;
LENGTH _OLDV_ $ 200;
ARRAY _TOCHAN_ _VAR;
DROP _FIRST OLDV1 _MODCNT _LINCNT _DOMOD ;
-FIRST = 0 ; _DOMOD = 0 ;
/*
The array _TOCHAN is critical to the default behavior. SAS
neatly provides the _CHARACTER_variable list for referring to all
the character variables within a step. The list can be overridden
by using the _VAR argument as mentioned above.
*/
DO OVER _TOCHAN_ ;
_OLDV_ = _TOCHAN_ ;
_TOCHAN_ = TRNWRD(_OLDV_,&_FROMSTR,
&_TOSTR);
%HOS _LOG_ON = YES %THEN %DO;
IF _TOCHAN_ NE _OLDV_ THEN DO;
IF _FIRST = 0 THEN _LINCNT + 1;
-FIRST = 1; _MODCNT+1; _DOMOD= 1;
PUT "NOTE: (CHGSTR) OBS=" _N_ " CHANGED "
_TOCHAN_=" FROM '" _OLDV_ " ' " ; END;
%END;
END;
END;
*/
There is a reason that all the data step variables local to this
macro have underscores in their names. It is extremely important
to try to avoid variable name collisions when writing macros that
crawl through many datasets. Using underscores will lessen the
chance of problems caused by coincidental encounters
*/
IF _DOMOD THEN REPLACE;
%HOS _LOG_ON = YES %THEN
IF DONE AND (_LINCNT >0) THEN PUT
"NOTE: (CHGSTR) FOR DATASET &_DS MODIFIED "
_MODCNT " STRINGS IN " _LINCNT " OBS.";
RUN;
%MEND;
*/
Part 2 MACRO TRUPCASE – while small, this macro will prove to
be useful within the main LIBTRAN routine as well as in
SOLUTION #3 to follow
when the code is retrieved for processing. Evaluation waits until after the end of expressions unresolved within single quotes, the macro would fail in a flurry of syntax errors. Because resolved while compiling the data step and the CALL wrap the string containing CALL EXECUTE SASHELP solution the same metadata is reached via views stored in the system. The this Innocent-looking details, such as the concatenation of spaces in the WHERE clause building code, enable the globalization of the system. The INDEX function is used to determine whether a dataset name or Libname matches against arguments. The extra space added to the end of both string guards against accidentally datasets in every known library except SASHELP and SASUSER. If specific lists of datasets and/or LIBREFs are specified using the optional arguments, then these must be space delimited. But lists of SAS character Variables can be described using standard SAS conventions...(for example: ID1-ID5)

/*
%MACRO TRUPCASE(ARG);TRIM(LEFT(UPCASE(&ARG)));%MEND;
/*

Part 3 MACRO LIBTRAN – The parent macro cascades the previously defined CHGSTR macro across a list of datasets in a series of libraries. The default action is to run across all existing datasets in every known library except SASHELP and SASUSER. If specific lists of datasets and/or LIBREFs are specified using the optional arguments, then these must be space delimited. But lists of SAS character Variables can be described using standard SAS conventions...(for example: ID1-ID5)

/*
%MACRO LIBTRAN( LIB=_ALL_ , DS=_ALL_ , FROMSTR= "NOTSPECIFIED" , TOSTR= "", VAR= _CHARACTER_ , LOG= YES );
%IF !(&LIB) EQ "" %THEN %DO;
%PUT;
%PUT ERROR: LIBTRAN MACRO NEEDS A LIB= ARGUMENT.;
%PUT; %END;
%ELSE %DO;
%LET WHRSTR= 1=1;
%END;
/*

Innocent-looking details, such as the concatenation of spaces in this WHERE clause building code, enable the globalization of the system. The INDEX function is used to determine whether a dataset name or Libname matches against arguments. The extra space added to the end of both string guards against accidentally matching an embedded substring.

/*
%IF NOT (&LIB = _ALL_ ) %THEN %DO;
%LET WHRSTR= INDEX(%TRUPCASE("&LIB")|| ' ', %TRUPCASE(LIBNAME)|| ' ') > 0 ;
%END;
%IF !(&DS) NE "" AND NOT (&DS= _ALL_ ) %THEN %DO;
%LET WHRSTR = &WHRSTR AND INDEX(%TRUPCASE(&DS))|| ' ', %TRUPCASE(MEMNAME)|| ' ') > 0 ;
/*

While the first example used the dictionary tables directly, in this solution the same metadata is reached via views stored in the SASHELP library. Using a data step allows a one step launch of CALL EXECUTE code based on the contents of the LIBNAME and MEMNAME columns/variables.

/*
DATA _NULL_; SET SASHELP.VTABLE
( WHERE=(&WHRSTR AND MEMTYPE="DATA") );
IF TRIM(LEFT(LIBNAME)) IN ("SASUSER","SASHELP")
THEN DELETE;
/*

Previously we mentioned that the type of string delimiter could be very important when using CALL EXECUTE to launch macro code. Here is the perfect example. If double quotes were used to wrap the string containing "%CHGSTR", then CHGSTR would be resolved while compiling the data step and the CALL EXECUTE would fail in a flurry of syntax errors. Because SAS leaves macro expressions unresolved within single quotes, the macro evaluation waits until after the end of the DATA_NULL step when the code is retrieved for processing.

/*
CALL EXECUTE('%CHGSTR( _DS_=' || %TRUPCASE(LIBNAME)|| ' "" || %TRUPCASE(MEMNAME)|| ' "" || _FROMSTR=4FROMSTR', || _TOSTR=_TOSTR || '"_VAR&_VAR, _LOG_ON= &LOG)');
*/

RUN;
%MEND;

IMPROVEMENTS AND SIMILAR SOLUTIONS

The LIBTRAN system contains the most versatile framework of the three example solutions in this paper. Similar routines can be used to launch macros that potentially affect every variable in every dataset or view, every index and every member in any or all catalogs...in every library known to the current session! The authors have created a variation that optionally acts as a global FIND string operation with user specified WHERE conditions. In addition, the macros have been enhanced to respect true word boundaries within strings (in a similar method to the SCAN function). In the interactive SAS world, another variation of these macros is tied to a menu bar item and uses a macro window to ask for confirmation before each replace operation.

SAS Version 8, introduced new functions and calls such as CALL RXCHANGE, CALL RXFREE, CALL RXSUBSTR, RXMATCH, and RXPARSE. The new routines can be used to enhance the LIBTRAN macro to allow pattern matching. The additions would allow powerful global find and replace based on regular expressions. In addition, Version 8 now offers auditing abilities that could conceivably act as the basis of an UNDO feature for global replacements.

SOLUTION #3

FINDING MISSING LEVELS OF IDs ACROSS MULTIPLE DATASETS AND LIBRARIES

THE INSPIRATION

While the previous two solutions were inspired by familiar system utilities, this solution evolved from a task routinely performed during Clinical Trial Case Report Form data management. Sections of Case Report Data are often saved as separate SAS Datasets. (In our examples, we have used mock datasets called DEMOG, DIARY and EFFICACY) Depending on the design and objective of the study, certain standard CRF sections may or not be present within the SAS libraries that comprise the study data. We designed a macro that, given a set of ID variables such as CLINIC, PATNUM and/or VISIT, will identify all datasets within all libraries that contain the set of ID variables and then present observation counts for every existing combination of IDs. Because this type of search is often performed to check for missing page/section counts, we added an option that only lists rows for which a level of ID combination is missing in at least one dataset. Beyond clinical data, this macro is useful for reporting on any set of SAS datasets that share a set of variables that act as ID or KEYS. If data were imported or entered in a system where the entry of the ID variables was not validated, the macro’s reports will instantly reveal where the IDs may have been entered incorrectly. This can be used to find KEYS that are missing within datasets.

EXAMPLE
Table 3 at the end of the paper shows example output from sample executions of this macro.

We generated the sample with the following calls

IDSTATUS( ID = CLINIC PATNUM);
IDSTATUS( ID= CLINIC PATNUM, LIB= MASTERLB,
REPTYPE= MISSING)

CALLING SYNTAX

IDSTATUS is called using the following arguments:

LIB= a list of LIBREFs, default is to scan ALL libraries
INCLUDE= A list of datasets, default is to use all datasets not excluded
EXCLUDE= A list of datasets to exclude, by default no dataset is explicitly excluded
ID= a list of ID variables. This argument is not optional
REPTYPE= Style of report - either COUNTS which is a
frequent count for every level of IDS in all datasets processed or MISSING, which is the same report with only rows contains a column that has missing value. Default is COUNTS - See TABLE 3 for an example.

**COMMENTED SOURCE CODE FOR MACRO IDSTATUS**

/*
Like the macro LIBTRAN, the IDSTATUS macro accepts a single library name or a list of LIBREFs. In addition, this macro will optionally support either a list of datasets to include in processing or a list to exclude. So the macro begins with a series of checks for error conditions such as specifying mutually exclusive arguments. (Note: This macro requires the TRUPCASE macro used in the previous section)
*/

%MACRO IDSTATUS( LIB=_ALL_ , INCLUDE=_ALL_ , EXCLUDE= _NONE_ , ID= _NONE_, REPTYPE = COUNTS );
%IF "&LIB" EQ "" %THEN %DO;
    %PUT;
    %PUT ERROR: IDSTATUS MACRO REQUIRES A COMPLETE LIB= ARGUMENT.;
    %PUT;%END;
%ELSE %IF NOT (( &INCLUDE= _ALL_ ) OR ( &EXCLUDE= _NONE_ ) ) %THEN %DO;
    %PUT;
    %PUT ERROR: IDSTATUS MACRO - 'INCLUDE' AND 'EXCLUDE' OPTIONS ARE MUTUALLY EXCLUSIVE.
    PLEASE CHOOSE ONE ONLY;
    %PUT;%END;
%ELSE %IF NOT ( &REPTYPE= COUNTS OR &REPTYPE= MISSING ) %THEN %DO;
    %PUT;
    %PUT ERROR: IDSTATUS MACRO - REPTYPE= &REPTYPE IS AN UNKNOWN REPORT, CHOOSE 'COUNTS' OR 'MISSING';
    %PUT;%END;
%ELSE %IF &ID= _NONE_ OR "&ID" = " " %THEN %DO;
    %PUT;
    %PUT ERROR: IDSTATUS MACRO - NO ID VARIABLES WERE SPECIFIED, USE 'ID OPTION LIKE ID= VAR1 VAR2';
    %PUT;
    %END;
%END;

/*
With error checking completed, the real works starts with a section devoted to building a WHERE clause. The code should be familiar after reading the last example…with the variation that we act on "ID" variables and an INCLUDE list of datasets now. Excluded dataset lists will be processed later. The strange little section of code that kicks off the WHERE clause, "=1", is inserted just so that we don’t have to work too hard to calculate when to use an "AND" in our expression.
*/
%IF _NOB = 1 THEN DO;
    VARLIST = COMPBL("&ID");
    NLIST = LENGTH(VARLIST);
    VARLIST= TRANSLATE(SUBSTR(VARLIST,1, NLIST), ",", " ");
    CALL SYMPUT("_TABID", TRIM(TRANSLATE(VARLIST, ",","")));
    END;
%ELSE %DO;
    %LET WHRSTR = &WHRSTR 1-1;
    %IF NOT ( &LIB= _ALL_ ) %THEN %DO;
    %LET WHRSTR = INDEX(%TRUPCASE("&LIB") || " ", %TRUPCASE(LIBNAME))|| " ", 0 ;
    %END;
    %IF "&INCLUDE" NE "" AND NOT ( &INCLUDE= _ALL_ ) %THEN %DO;
    %LET WHRSTR = &WHRSTR AND INDEX(%TRUPCASE("&INCLUDE") || " ", %TRUPCASE(NAME)) || " ", 0 ;
    %END;
    %END;

/*
The system table SASHELP.VCOLUMN details every variable known to exist within a dataset. We will use it to search for datasets that contain the ID variables that have been specified. First we trim the list by way of our where clause and exclude lists.. Sharp-eyed readers will understand that this macro fails when datasets exist with the name of "_NONE_"
*/

DATA _IDTASK_;
SET SASHELP.VCOLUMN( WHERE=(&WHRSTR AND MEMTYPE="DATA" ));
IF TRIM(LEFT(LIBNAME)) IN ("SASUSER","SASHELP") THEN DELETE;
IF INDEX(%TRUPCASE("&EXCLUDE") || " ", %TRUPCASE(MEMNAME)) || " ", 0 THEN DELETE;
KEEP LIBNAME MEMNAME;
RUN;

/*/ Depending on whether there is more than one ID variable or not, there will be multiple observations in the table we just built containing the name of datasets that need to be processed. */

PROC SORT NODUPKEY ; BY LIBNAME MEMNAME ;

/*/ At least as far as this paper is concerned, what follows is the mother of all code-writing data steps! It takes information gleaned from dictionary tables and uses both CALL EXECUTE and SYMPUT in concert to build reports. */

DATA _NULL_;
LENGTH VARLIST $ 200;
RETAIN VARLIST;
SET _IDTASK_ END= _DONE_ ;
_NOB + 1;
*/

Lists of ID variables come into the macro using space delimiters. But we also need the list delimited by commas for a PROC SQL step that we are writing, and then also delimited by asterisks for a PROC TABULATE step used for reporting. Hence the following code… The SYMPUT statements reserves the TABULATE code fragment as macro for later substitution after the current step ends */

IF _NOB = 1 THEN DO;
    VARLIST = COMPBL("&ID");
    NLIST = LENGTH(VARLIST);
    VARLIST= TRANSLATE(SUBSTR(VARLIST,1, NLIST), ",", " ");
    CALL SYMPUT("_TABID", TRIM(TRANSLATE(VARLIST, ",","")));
    END;
*/

If you find this style of macro coding to be too involved to follow, here are some hints: First: request a version from the authors that has not been formatted in columns (!) More importantly: try running the code with the SAS option MPRINT on and the work backwards from the code that is generated. That "vision thing" is important for programmers as well as macros: You must be able
to visualize how and when SAS parses, resolves, compiles and processes macros and BASE code to write macros like these.

*/
CALL EXECUTE('PROC SQL NOPRINT;CREATE TABLE _IDINT AS ');
CALL EXECUTE('SELECT COUNT(*) AS OBSCNT , "'||LIBNAME
   ||' " AS LIBREF ,"'||MEMNAME ||' " AS DATASET,
   |" GROUP BY | VARLIST | "
;);
CALL EXECUTE('PROC APPEND DATA=_IDINT
   OUT=_IDSTATS;RUN;');
IF _DONE_ THEN
   CALL SYMPUT( "_IDNDS", PUT( _NOB , 5 . )
;)
RUN;
PROC SORT DATA=_IDSTATS ; B Y &ID;
/*
Make that two mothers…at least when a "MISSING" style report
is being generated. This optional step counts up the number of
datasets that each level of ID combination appears in and merges
it with the row of observation counts. If the count of datasets
equals the total number of datasets, then we deduce that the ID
level is present in each dataset and the row should not be in the
missing level report.
*/
DATA _NULL_; %IF &REPTYPE = MISSING %THEN %DO;
   CALL EXECUTE('PROC FREQ NOPRINT; TABLE &_TABID
   /OUT=_IDFREQ;RUN;');
   CALL EXECUTE('DATA _IDSTATS; MERGE _IDSTATS _IDFREQ;
   BY &ID; IF COUNT ="|"&_IDNDS" |
   |" THEN DELETE ;
   );
%END;
CALL EXECUTE('PROC TABULATE DATA=_IDSTATS
   ;CLASS LIBREF DATASET &ID ; FREQ OBSCNT;
   'BOX="|&ID| INDENT=5; KEYLABEL N=" |RUN;');
RUN;
/*
The last step is to clean up any temporary datasets that we
created and then we are outta here...
*/
PROC DATASETS LIBRARY=WORK NOWARN NOLIST;
DELETE _IDSTATS ; DELETE _IDFREQ;
DELETE _IDINT ; RUN;
%MEND;

IMPROVEMENTS AND SIMILAR SOLUTIONS
Some sort of warning should be generated when a dataset
contains some, but not all, of the ID variables that are to be
processed. In addition, the list of ID levels missing in a dataset
should contain all datasets present. We have included both of
these improvements in an extended version of the IDSTATUS
macro. Here we pared down the code in order to reduce the
complexity of an example that is already quite elaborate.

Nonetheless, the macro works quite well for the amount of
specification given to it…

CONCLUSION
We have demonstrated three examples of code writing tools that
“look” into the existing SAS metadata to retrieve dictionary item
names and properties that are then used as building blocks. We
believe that MACROS based on dictionary tables (and CALL
EXECUTE) to be the most “dynamic” BASE SAS routines that can
be written. These techniques have been used for many years, yet
the tools you construct using dictionary components can be
among the most versatile routines in your library of reusable code.
We have also used these methods within SCL and within involved
language parsing routines. And we “envision” many useful
variants that are just asking to be written. More on these later…

REFERENCES


ABOUT THE AUTHORS
Michael E. Tomb is the founder of INFORMATICA ECS, INC. a software development and consulting
compny. He is an advanced SAS certified professional with over 20 years experience using the SAS system across several
operating systems. In addition, Mr. Tomb has been the designer
and lead programmer of several commercially available scientific applications for the chemical and pharmaceutical industries.

James R. Carter is a SAS certified professional and has been working with SAS software for over 13
years. His SAS experience ranges from data management and statistical modeling to AF application development. He has been
a repeat presenter at NESUG conferences and occasional lurking
contributor on the SAS-L. When not busily writing SAS code, he
likes to relax by juggling the occasional flaming torch.

CONTACT INFORMATION
To receive updated versions of these macros or for any
questions, contact the authors using:

Michael E. Tomb
Informatica ECS, Inc.
PO Box 39398
Rochester, NY 14604
Phone: 585.256.0401 Fax: 585.271.8746
Email: info@informaticaecs.com
Web: www.informaticaecs.com

SAS® and all other SAS Institute Inc. product or service names
are registered trademarks or trademarks of SAS Institute Inc. in
the USA and other countries. ® indicates USA registration.
Other brand and product names are registered trademarks or
trademarks of their respective companies.
Table 1.
Sample Output from Macro COMPLIBS

DATASETS TO BE COMPARED WITHIN LIBRARIES MASTERLB AND UPDATELB

<table>
<thead>
<tr>
<th>Ob</th>
<th>Dataset</th>
<th>Last Date Modified in</th>
<th>Last Date Modified in</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEMOG</td>
<td>01FEB01:20:06:46</td>
<td>08FEB01:11:28:43</td>
</tr>
<tr>
<td>2</td>
<td>EFFICACY</td>
<td>01FEB01:08:54:13</td>
<td>08FEB01:11:28:41</td>
</tr>
</tbody>
</table>

COMPARISON - DATASET DEMOG IN LIBRARIES MASTERLB AND UPDATELB
FOR DATASETS MODIFIED AFTER 08FEB01:00:00:00

COMPARE Procedure
Comparison of MASTERLB.DEMOG with UPDATELB.DEMOG
(Method=EXACT)

Variables with Unequal Values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Ndif</th>
<th>MaxDif</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEX</td>
<td>CHAR</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>CHAR</td>
<td>32</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>AGE</td>
<td>NUM</td>
<td>8</td>
<td>1</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Value Comparison Results for Variables

<table>
<thead>
<tr>
<th>Ob</th>
<th>Base Value</th>
<th>Compare Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>M</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ob</th>
<th>Base Value</th>
<th>Compare Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>HOWL E. U.</td>
<td>HOWLY E. U.</td>
</tr>
<tr>
<td>51</td>
<td>ARTICHOKE A. Q.</td>
<td>FENNEL A. Q.</td>
</tr>
<tr>
<td>64</td>
<td>DARKBLOOM A. I.</td>
<td>NABOKOV A. I.</td>
</tr>
</tbody>
</table>

Table 2.
Sample LOG Report from Macro LIBTRAN

NOTE: (CHGSTR) OBS=7 CHANGED NAME=JAVERT H. M. FROM 'VALJEAN H. M.'
NOTE: (CHGSTR) OBS=50 CHANGED NAME=JAVERT I. J. FROM 'VALJEAN I. J.'
NOTE: (CHGSTR) FOR DATASET MASTERLB.DEMOG MODIFIED 2 STRINGS IN 2 OBS.
NOTE: There were 104 observations read from the data set MASTERLB.DEMOG.
NOTE: The data set MASTERLB.DEMOG has been updated. There were 2 observations rewritten.
Table 3. Sample Reports from Macro IDSTATUS

**FULL IDSTATUS REPORT**

<table>
<thead>
<tr>
<th>CLINIC PATNUM</th>
<th>LIBREF</th>
<th>MASTERLB</th>
<th>UPDATELB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DEMOG</td>
<td>DIARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DEMOG</td>
<td>DIARY</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>1 13 1</td>
<td>1 13 1</td>
</tr>
<tr>
<td>A01968</td>
<td></td>
<td>1 15 .</td>
<td>1 15 1</td>
</tr>
<tr>
<td>A30288</td>
<td></td>
<td>1 5 1</td>
<td>1 5 1</td>
</tr>
<tr>
<td>A31272</td>
<td></td>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>B40489</td>
<td></td>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>C50450</td>
<td></td>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>1 2 1</td>
<td>1 2 1</td>
</tr>
<tr>
<td>A13200</td>
<td></td>
<td>1 14 1</td>
<td>1 14 1</td>
</tr>
<tr>
<td>B02689</td>
<td></td>
<td>1 3 1</td>
<td>1 3 1</td>
</tr>
<tr>
<td>B02713</td>
<td></td>
<td>1 3 1</td>
<td>1 3 1</td>
</tr>
</tbody>
</table>

**IDSTATUS MISSING PAGES (LEVELS) REPORT**

<table>
<thead>
<tr>
<th>CLINIC PATNUM</th>
<th>LIBREF</th>
<th>MASTERLB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DEMOG</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>1 15 .</td>
</tr>
<tr>
<td>A30288</td>
<td></td>
<td>1 . 1</td>
</tr>
<tr>
<td>D20163</td>
<td></td>
<td>1 . 1</td>
</tr>
<tr>
<td>M12444</td>
<td></td>
<td>1 . 1</td>
</tr>
<tr>
<td>S32778</td>
<td></td>
<td>1 8 .</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>1 5 .</td>
</tr>
<tr>
<td>D14811</td>
<td></td>
<td>1 . 1</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>1 . 1</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>1 . 1</td>
</tr>
<tr>
<td>F48389</td>
<td></td>
<td>1 . 1</td>
</tr>
<tr>
<td>N49357</td>
<td></td>
<td>1 2 .</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td>30 192 7</td>
</tr>
</tbody>
</table>