Paper AD-006

Resolving Value Differences among Duplicates, A Rule-based Approach

David H. Abbott, L. Douglas Melton, Steven C. Grambow, and George L. Jackson
Veterans Affairs Health Services Research & Development
Durham, NC

ABSTRACT

When cleaning data, analysts may encounter sets of observations with the same value of the intended primary key (e.g., Social Security Number). To proceed, the analyst must resolve these into a single observation. Total duplicates can simply be discarded, but when variable values differ within the duplicate observations, the task is more complex. Decisions have to be made about which *values* to keep and which to discard.

Using Base SAS® 9.1, we have created a macro, DiffResolve, to aid analysts in listing the value differences and writing and enacting rules to resolve the differences. This macro is valuable for smaller projects not wishing to invest in software products that support data cleaning in general, such as the SAS® Data Quality Solution.

The macro is suited for use by intermediate users of SAS. Advanced users may find the code of interest. For example, the rules used to resolve value differences are represented in a SAS dataset rather than in SAS code. The macro code fetches the rules and applies them in turn in the proper execution context.

INTRODUCTION

In some projects, it is not unusual to encounter duplicate observations in SAS datasets. These come in two types: exact duplicates and key duplicates. An **exact duplicate** matches some other observation(s) exactly (i.e. the values of all variables match). A **key duplicate** is an observation whose primary key (e.g. Social Security Number) matches the primary key of some other observation(s) in the SAS dataset but whose other values do not fully match. Exact duplicates are easily removed, but not so key duplicates. When key duplicates are present, the analyst must resolve all variable value differences among matching key duplicates and produce a dataset with a single observation for each primary key value.

A given project may have tens or even hundreds of key duplicates and each observation may have hundreds of variables, so the total number of value differences to be resolved can easily be in the thousands. Attempting a purely manual resolution of the differences can be a cumbersome and exceedingly tedious task! Incorporating some form of software assistance with this task is critical to accurately resolve discrepancies in a time efficient manner. Papers presented at past SAS® Global Forums have discussed facilities in SAS to help identify and segregate both exact duplicates and key duplicates (see references), but provide little help for resolving the value differences among key duplicates. Existing software tools, such as the SAS Data Quality Solution, are available to address this need and represent a good solution for large, well-funded projects. However, smaller projects often need a less complicated, less costly, and more transparent solution.

Base SAS, augmented with the macro facility, is a powerful programming environment that allows the development of reusable and sharable solutions to many problems. This paper describes how we have used this environment to create a SAS macro, DiffResolve, that helps the analyst to resolve the value differences among key duplicates.

This macro was initially developed to resolve discrepancies in a project collecting healthcare data from multiple sites within a single healthcare system, a relatively common situation in health services research. The Department of Veterans Affairs conducted a medical chart abstraction to explore the quality of colorectal cancer treatment among a sample of patients from across the VA. Chart abstractions were conducted at each participating medical facility. Because some patients received treatment at multiple facilities, they had multiple entries in the data, which resulted in 146 key duplicates among the 3047 records in the original dataset received for cleaning. Each observation was comprised of 253 variables, approximately 11% of which differed in value among the key duplicates, yielding 4047 differences to be resolved.

UNDERSTANDING THE PROBLEM

Here is an example of a simple dataset composed of 3 sets of key duplicates …

Obs	State	Capitol	LargestCity	Area	Population
1	AL	Montgomery	Birmingham	52.4	5.6
2	AL	?	Birmingham	.	5.6
3	AR	LittleRock	LittleRock	53.2	-2.6
4	AR	LittleRock		53.2	2.8
5	FL			.	18.1
6	FL	Tallahassee		65.8	0.0
7	FL	Tallahassee	Jacksonville	65.8	.

Observations [1,2], [3,4], and [5,6,7] are the three sets of key duplicates corresponding to primary key values AL, AR, and FL, respectively. Let's suppose these seven records were extracted from a larger dataset intended to have one observation for each state (the primary key). In such case, the larger dataset would have had 47 non-duplicate observations (one for each of the remaining states) plus the 7 observations shown above. See [Kohli] for information on how to split a given dataset into two datasets, one with no key duplicates and the other composed of the key duplicates.

In this simple example, just three rules are required to resolve the value conflicts:

- Keep the non-missing value (applies to all variables).
- Discard values zero or less (applies to population).
- Discard values containing non-alphabetic characters (applies to Capitol).

Here is the dataset that results after differences have been resolved and key duplicates eliminated:

Obs	State	Capitol	LargestCity	Area	Population
1	AL	Montgomery	Birmingham	52.4	5.6
2	AR	LittleRock	LittleRock	53.2	2.8
3	FL	Tallahassee	Jacksonville	65.8	18.1

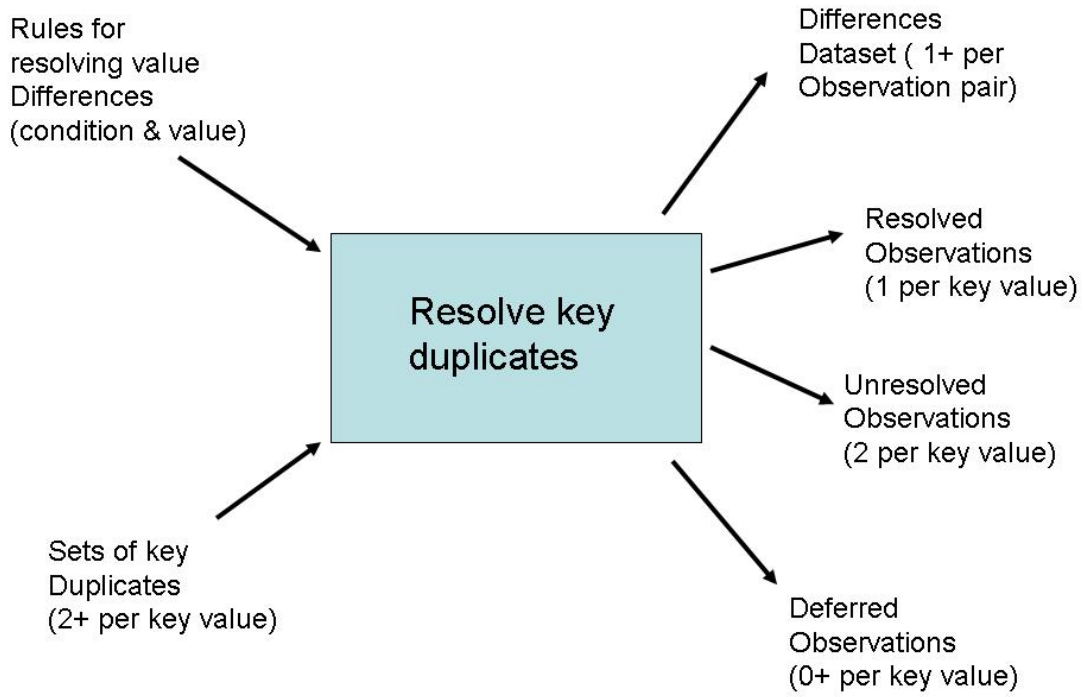
ANALYZING THE TASK

With or without software tool support, the component tasks of dealing with key duplicates are the same:

- Create a dataset that contains just the sets of key duplicates and no other observations.
- Identify the field value differences that occur within each set of key duplicates and capture these in a form convenient for viewing and further analysis, in other words, in a Differences Dataset.
- Identify rules for resolving the value differences. One broadly useful rule was illustrated above: if one value of a variable is a missing then use the other value. Another rule that might apply is: if the variable is binary use the 1 and not the 0 (e.g. an indication of occurrence is stronger evidence than non-occurrence). One rule that can always be applied, as a last resort, is to simply set the variable to missing.
- Use the rules to resolve as many differences as possible. If the number of key duplicates for a given key is 2 then this is straight-forward. The case of 3 or more in the duplicate group is more complicated. The simplest approach to handling this situation is iterative: combine the first and last observation in each group into a single observation on each pass and set aside the intervening observations for processing in a subsequent pass. Continue until no observations need be set aside.
- Segregate the resulting observations into three datasets: 1) fully resolved observa.tions, 2) observations with some unresolved differences remaining (and requiring manual processing), and 3) observations to be processed in a later iteration of this process, derived from keys with 3 or more key duplicates (see point above).
- After completing any manual resolution and/or iteration required, add the full set of observations derived from the key duplicates back into the dataset from which they were extracted.

The inputs and outputs associated with these tasks are illustrated in the following context diagram:

Context Diagram for Resolving Value Differences Among Key Duplicates



USING THE DIFFRESOLVE MACRO

Here's the signature of the DiffResolve macro :

```
%MACRO DiffResolve(

 /*Input Datasets*/
  DupSetsDs=, /*Contains sets of key duplicates to be resolved*/
  RulesDs=, /*Each obs is a rule for resolving value differences*/

 /*Output Datasets*/
  DifferencesDs=dr_diffs, /*each obs is an unresolved value difference*/
  FixedDs=dr_fixedObs, /*obs with all value differences resolved*/
  FailedDs=dr_failedObs, /*obs with unresolved differences*/
  DeferredDs=dr_deferObs, /*obs to be processed in later iterations*/

 /*Other*/
  PrimaryKey=, /*Variable intended to be primary key of the obs*/
  MaxVars=400, /*max number of variables allowed in an obs*/
  MaxVarLength=80, /*max allowed length of character variables*/
  CleanUp=1, /*use 0 if want to examine internally generated datasets*/
  Sp=tmp /*Scratch prefix, i.e. name prefix used for macro internal items*/
);
```

Some comments on this signature include:

- The dataset (…Ds) parameters follow directly from the context diagram.
- The PrimaryKey parameter is currently restricted to being a single variable; this is often the case and, if not, a single variable can be added to the dataset that is one-to-one with the multi-variable key.
- The remaining parameters in the "Other" group are implementation related and defaulted to values that will work for most users most of the time.

To help make these notions more palpable, here are sample datasets:

DupSetDs
See the first dataset in "Understanding the Problem" above, as it is, in fact, a dataset comprised of 3 sets of key duplicates.

RuleDs
This dataset lists the rules cited above cast into the form required for this macro:

Obs	pattern	assign	type
1	missing(@anyCharVar(A))	@anyCharVar(B)	C
2	missing(@anyNumVar(A))	@anyNumVar(B)	N
3	NOTALPHA(@capitol(A))	@capitol(B)	C
4	@area(A) le 0	@area(B)	N
5	@population(A) le 0	@population(B)	N

The "@" symbols mark the start of a variable name (or wild card that stands in for same). The "A" and the "B" just stand for "the one" and "the other" of the two conflicting values, whichever they may be. Using this artifice, the rules don't care which of the two duplicates is encountered first and one such rule is equivalent to two order specific rules. SAS Base doesn't like mixing character and numeric data, so it works best just to make rules specific to one type or the other.

FixedDs
See the second dataset in "Understanding the Problem" above.

FailedDs
For this example, the FailedDs dataset is empty. The observations, if present, would look the same as for FixedDs, but would, of course, either require new rules or manual intervention for resolution.

DeferredDs
This example has a single deferred observation:

Obs	State	Capitol	LargestCity	Area	Population
1	FL	Tallahassee		53.5	0

DiffResolve resolves the first and last observation in each set of duplicates and defers any observations occurring between the two - in this case, the second duplicate for FL – to a subsequent iteration of processing.

DifferencesDs
The differences that the macro identifies are captured in this dataset.

Obs	First Record Value	Current Record Value	Primary Key: State	Dupli-cate Num.	Variable Name
1	52.4	.	AL	2	Area
2	.	65.8	FL	2	Area
3	.	65.8	FL	3	Area

4	Montgomery	?	AL	2	Capitol
5		Tallahassee	FL	2	Capitol
6		Tallahassee	FL	3	Capitol
7	LittleRock		AR	2	LargestCity
8		Jacksonville	FL	3	LargestCity
9	-2.6	2.8	AR	2	Population
10	18.1	0	FL	2	Population
11	18.1	.	FL	3	Population

By studying this dataset the analyst is able to infer the rules required to resolve the differences. Looking at this dataset, you can quickly see that:

- All the differences in **Area** can be resolved by keeping the non-missing value.
- Resolving the **Capitol** differences also requires the alpha characters only rule.
- Resolving the **Population** differences requires the rule to reject numeric value <= 0.

DESIGNING THE MACRO

Some key issues in the design of this macro include:

- Handling three or more duplicates – Part of the strategy adopted was described above: route any observation that is not FIRST or LAST in its duplicate set to the DeferredDs dataset. But should the DifferencesDs include differences detected in these observations? We think yes since the analyst presumably benefits from a comprehensive view of the differences. When identifying differences each observation in a set is compared to the FIRST observation, whether it will be output to DeferredDs or not.
- Representing the three pertinent observations – The three are: the current observation, the first observation in the key duplicate set, and the combined observation that is being built from these two. The SAS data step enables access to the current observation by default, but simultaneously accessing the variables from the FIRST observation and the observation being built requires some artifice to be used. We chose to use three arrays of retained variables – one per pertinent observation. Surprisingly, it works best to associate the variables in the current observation to the array representing the combined observation since otherwise the length and format information for each variable is difficult to carry forward.
- Character vs. numeric – Actually, the macro requires two arrays per pertinent observation since SAS® does not allow mixing character and numeric variables in the same array. This restriction complicates the logic of the macro, but still, we liked it better than adding prefixes or suffixes to variable names to store additional observations since this alternative approach invariably constrains the variable names that can be present in DupSetDs.
- Applying the difference resolution rules – This task can be viewed in three steps. First, RuleDs is converted in a more implementation friendly rep (&sp.RuleSpecsDs)The main work here is changing the variable names to data array indices and creating two versions of each rule – one that considers the first observation in a duplicate group primary and another that considers the last primary. Second, for each occurrence of a difference, the logic scans &sp.RuleSpecsDs to find a rule that applies. Third, the rule is enacted with the help of %unquote which causes the macro processor to process an expression a second time (required for the variable wild-carding to work out). See the helper macro, ResolveVar, at the end of the appendix.
- Avoiding name clashes – The macro needs to use various variable names but must not use names that collide with the variable names in the DupSetDs. Nor should any of the several temporary datasets that it uses collide with any other temporary datasets in the user's SAS session. By default, the macro prefixes all of its names with "tmp". Further, the macro allows the user to specify an alternative string, say "dr_", if "tmp" is for some reason unsatisfactory (e.g. the invoking context uses datasets of form tmp.* and these datasets must be unaltered across an invocation of DiffResolve).

IMPLEMENTATION

The code for the DiffResolve macro is exhibited in the Appendix. It was written and tested in a Microsoft Windows environment, but should work for any operating system that Base SAS supports.

CONCLUSION

The DiffResolve macro is an efficient method for creating a single observation that captures the appropriate values for variables that originally had different values for a given set of duplicate observations. From a statistical perspective, DiffResolve macro prevents unnecessary data loss since it pulls together information from a group of duplicate observations that otherwise would need to be discarded due to the value differences. Even if the analyst arbitrarily chooses to retain only the first observation in the set of duplicates, useful data will likely still be lost. For instance, say there are two observations that only differ by one binary variable. In the first duplicate the binary variable is missing (=.N) and in the second duplicate line the value of the binary variable is 1. Picking the first observation and simply discarding the second means the binary variable to remains missing (=.N) thus adding no information to the analysis. However, with the suggested macro the value of 1 in the second duplicate observation replaces the missing value in the first duplicate with a 1. In short, the provided SAS MACRO removes duplicates and pulls the best value out of a set of conflicting values and places the value in the unified observation that is retained.

REFERENCES

Peterson, Brett J. 2006. "**Finding a Duplicate in a Haystack**." Proceedings of the 31st Annual SAS Users Group International Conference, San Francisco CA, http://www2.sas.com/proceedings/sugi31/164-31.pdf.

Kohli, Monal 2006. "**Project Duplication: Eradication Techniques**." Proceedings of the 31st Annual SAS Users Group International Conference, San Francisco CA, http://www2.sas.com/proceedings/sugi31/031-31.pdf.

Sienersm , Vicki and Nash, Brian 2000. "**Removing Duplicates: PROC SQL Can Help You 'See'**." Proceedings of the 25th Annual SAS Users Group International Conference, Indianapolis IN, http://www2.sas.com/proceedings/sugi25/25/cc/25p106.pdf.

ACKNOWLEDGMENTS

The views expressed in this paper are those of the authors and do not necessarily reflect the position or policy of the Department of Veterans Affairs or the United States government.

Dr. Jackson is supported by a Merit Review Entry Program (MREP) award from the Department of Veterans Affairs Health Services Research & Development Service (VA grant MRP 05-312). Mr. Melton is supported by a National Research Service Award-Agency for Healthcare Research and Quality Pre-Doctoral Fellowship intuitional training grant to the University of North Carolina at Chapel Hill.

Without the leadership and encouragement of Dr. Dawn Provenzale, director of the Durham Epidemiologic Research and Information Center at the Durham VA Medical Center, this work could not have occurred. She takes a strong interest in fostering many dimensions of excellence in her employees.

CONTACT INFORMATION

Name	David H. Abbott
Enterprise	Center for Health Services Research in Primary Care
Address	Durham Veterans Affairs Medical Center
	HSR&D Service (152)
	508 Fulton St.
City, State ZIP	Durham, NC 27705
Work Phone:	919-668-7205
E-mail:	david.abbott@va.gov

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 trademarks of their respective companies.

APPENDIX — SOURCE CODE FOR DIFFRESOLVE

```
%MACRO DiffResolve( /* Resolves differences in key duplicates */
  /*input datasets*/
    DupSetsDs=, /*contains sets of key duplicates, duplicates adjacent*/
    RulesDs=, /*contains rules for resolving value differences in duplicates*/
  /*output datasets*/
    FixedDs=dr_fixedObs, /*one obs for each pair of resolved duplicates*/
    FailedDs=dr_failedObs, /*composite obs for unresolvable duplicate pair*/
    DeferredDs=dr_deferObs, /* 3rd, 4th, etc. obs in a duplicate set */
    DifferencesDs=dr_Diffs,  /* enumerated diffences extant in DupSetDs */
  /*Other*/
    PrimaryKey=, /*the variable name of primary key for DupSetDs observations*/
    MaxVars=400, /* number of variables in DupSetDs must be less than this*/
    MaxVarLength=80, /* max allowed character variable length*/
    cleanUp=1, /* set to 0 if want to examine DiffResolve''s tmp datasets*/
    sp=tmp /*scratch prefix, DiffResolve is allowed to clobber anything named
               with this prefix, be it dataset, variable, or MACRO variable while
               doing its work. Set to something else short if prefer/need to*/
);

 %* Assemble meta data about &DupSetsDs and generate &sp.VarMapDs;
 ODS select none;
 PROC  CONTENTS data=&DupSetsDs;
 ODS output variables=&sp.VarListDs;
 RUN;
 ODS select all;
 PROC  SORT  data=&sp.VarListDs; by Num; RUN;
 DATA &sp.VarMapDs;
   retain charVars numVars origSeqVars cntChar cntNum; /*actual retain*/
   retain num variable type numIndex charIndex; /*variable order retain*/
   length charVars numVars origSeqVars $%eval(&MaxVars * 32);
   set &sp.VarListDs end=lastRec; * from PROC  CONTENTS ODS output;
   if type eq 'Num ' then do;
     numVars = trim(numVars) || ' ' || trim(put(variable, $32.));
     cntNum+1; numIndex=cntNum; charIndex=.;
   end;
   else if type eq 'Char ' then do;
     charVars = trim(charVars) || ' ' || trim(put(variable, $32.));
     cntChar+1; charIndex=cntChar; numIndex=.;
   end;
   else put '>> Hmmm ... neither num nor char';
   origSeqVars = trim(origSeqVars) || ' ' || trim(put(variable, $32.));
   if lastRec then do;
     call symput("&sp.allCharVars", trim(charVars));
     call symput("&sp.allNumVars", trim(numVars));
     call symput("&sp.origSeqVars", trim(origSeqVars));
     call symput("&sp.cntNum", left(put(cntNum, 4.)));
     call symput("&sp.cntChar", left(put(cntChar, 4.)));
   end;
   keep num variable type numIndex charIndex;
 RUN;

%* Generate the RuleSpecsDs from &RulesDs, for each rule:
    a. convert var names to integers,
    b. AnyChar/AnyNum to &index,
    c. duplicate for curRec vs. firstRec,
    d. determine target variable from pattern,
    e. write two elaborated rules;
DATA &sp.RuleSpecsDs (keep=varIndex pattern assign type);
  length varIndex $6 num1-num&&&sp.cntNum $32 char1-char&&&sp.cntChar $32;
  retain num1-num&&&sp.cntNum  char1-char&&&sp.cntChar ;
  array numVec{*} $ num1-num&&&sp.cntNum;
  array charVec{*} $ char1-char&&&sp.cntChar;
```

```
  if _N_ eq 1 then do;
    %* ingest &sp.VarMapDs;
    &sp.dsid=open("&sp.VarMapDs");
    do i = 1 to attrn(&sp.dsid,"nobs");
      set &sp.VarMapDs(rename=( type=typeIn));
      %* load up numVec and charVec with the variable names;
      if typeIn eq 'Char' then do;
        charVec(charIndex) = variable;
      end;
      else if typeIn eq 'Num' then do;
        numVec(numIndex) = variable;
      end;
      else put ">> Rule work unexpected: type neither C nor N";
    end;
  end;
  set &RulesDs(rename=(assign=assignIn pattern=patternIn type=typeIn ));

  posAt=indexc(patternIn, '@');
  tmp = scan( substr(patternIn, posAt+1), 1);
  if typeIn eq 'C' then do;
    do i=1 to &&&sp.cntChar;
      if lowcase(tmp) eq lowcase(charVec(i)) then leave;
    end;
    if i le &&&sp.cntChar then varIndex=put(i, 4.); /* limits to 9999 variables */
    else if tmp eq "anyCharVar" then varIndex='&index';
    else varIndex="bad";
    type='C';
    pattern=tranwrd(patternIn, '@' || trim(tmp) || "(A)",
       "curRecChar" || "(" || trim(varIndex) || ")");
    assign= tranwrd(assignIn, '@' || trim(tmp) || "(B)",
       "firstRecChar" || "(" || trim(varIndex) || ")");
    output;
    pattern=tranwrd(patternIn, '@' || trim(tmp) || "(A)",
       "firstRecChar" || "(" || trim(varIndex) || ")");
    assign= tranwrd(assignIn, '@' || trim(tmp) || "(B)",
       "curRecChar" || "(" || trim(varIndex) || ")");
    output;
  end;
  else if typeIn eq 'N' then do;
    do i=1 to &&&sp.cntNum;
      if lowcase(tmp) eq lowcase(numVec(i)) then leave;
    end;
    if i le &&&sp.cntNum then varIndex=put(i, 4.); /* limits to 9999 variables */
    else if tmp eq "anyNumVar" then varIndex='&index';
    else varIndex="bad";
    type='N';
    pattern=tranwrd(patternIn, '@' ||  trim(tmp) || "(A)",
       "curRecNum" || "(" || trim(varIndex) || ")");
    assign= tranwrd(assignIn, '@' ||  trim(tmp) || "(B)",
       "firstRecNum" || "(" || trim(varIndex) || ")");
    output;
    pattern=tranwrd(patternIn, '@' ||  trim(tmp) || "(A)",
       "firstRecNum" || "(" || trim(varIndex) || ")");
    assign= tranwrd(assignIn, '@' ||  trim(tmp) || "(B)",
       "curRecNum" || "(" || trim(varIndex) || ")");
    output;
  end;
  else put ">> Rule work unexpected: type neither C nor N";
RUN;

%* Process the user''s &DupSetDs
  Compare 2nd to 1st, 3rd to 1st etc and record differing fields
  Only fix 2nd to 1st differences (if fix code provided/active);

DATA &FixedDs(keep= &&&sp.allCharVars &&&sp.allNumVars)
     &DeferredDs(keep= &&&sp.allCharVars &&&sp.allNumVars)
     &FailedDs(keep= &&&sp.allCharVars &&&sp.allNumVars)
```

```
     &differencesDs(keep= &PrimaryKey &sp.recNum &sp.Name &sp.firstRecCharVal
         &sp.curRecCharVal);
   length &sp.numName1-&sp.numName&&&sp.cntNum
         &sp.charName1-&sp.charName&&&sp.cntChar $32 ;
   length &sp.prevChar1-&sp.prevChar&&&sp.cntChar
         &sp.curRecChar1-&sp.curRecChar&&&sp.cntChar
         &sp.firstRecCharVal  &sp.curRecCharVal $&MaxVarLength;
   retain &sp.prevNum1-&sp.prevNum&&&sp.cntNum &sp.prevChar1-&sp.prevChar&&&sp.cntChar
         &sp.curRecNum1-&sp.curRecNum&&&sp.cntNum &sp.curRecChar1-
&sp.curRecChar&&&sp.cntChar
         &sp.numName1-&sp.numName&&&sp.cntNum &sp.charName1-&sp.charName&&&sp.cntChar;
   set &DupSetsDs; by &PrimaryKey; /* must precede array stmts */
   array curRecChar{&&&sp.cntChar} $ &sp.curRecChar1-&sp.curRecChar&&&sp.cntChar;
   array curRecNum{&&&sp.cntNum} &sp.curRecNum1-&sp.curRecNum&&&sp.cntNum;
   array firstRecChar{&&&sp.cntChar}$ &sp.prevChar1-&sp.prevChar&&&sp.cntChar;
   array firstRecNum{&&&sp.cntNum} &sp.prevNum1-&sp.prevNum&&&sp.cntNum;
   array outRecChar{&&&sp.cntChar}$ &&&sp.allCharVars; *values for the current record;
   array outRecNum{&&&sp.cntNum} &&&sp.allNumVars;  *values for the current record;
   array varCharName{&&&sp.cntChar} $ &sp.charName1-&sp.charName&&&sp.cntChar;
   array varNumName{&&&sp.cntNum} $ &sp.numName1-&sp.numName&&&sp.cntNum;
   label &sp.firstRecCharVal="First Record Value"
         &sp.curRecCharVal="Current Record Value"
         &sp.recNum="Position of Record in DupSet";
   if _N_ eq 1 then do;
     %* Set up the vectors of char and num variable names;
     &sp.dsid=open("&DupSetsDs");
     iChar=0; iNum=0;
     do i = 1 to attrn(&sp.dsid,"nvars");
       &sp.typeInd = varType(&sp.dsid, i);
       if &sp.typeInd eq 'C' then do;
         iChar+1;
         varCharName(iChar) = varname(&sp.dsid,i);
       end;
       else if &sp.typeInd eq 'N' then do;
         iNum+1;
         varNumName(iNum) = varname(&sp.dsid,i);
       end;
       else put ">> unexpected: type neither C nor N";
     end;
   end;
   do i= 1 to &&&sp.cntChar; curRecChar(i)= outRecChar(i); end; /* yes, this is right*/
   do i= 1 to &&&sp.cntNum; curRecNum(i)= outRecNum(i); end; /* again, this is right */
   if first.&PrimaryKey then do;
     do i=1 to &&&sp.cntNum; firstRecNum(i) = curRecNum(i); end;
     do i=1 to &&&sp.cntChar; firstRecChar(i) = curRecChar(i); end;
     numPending =0;  &sp.recNum =1;
   end;
   else do; /*Not first.&PrimaryKey*/;
     &sp.recNum + 1;

     /*        Char variables         Char variables        ========== */
     do i=1 to &&&sp.cntChar;
       &sp.outRecCharVal=""; /*clear value from prev iteration*/
       if curRecChar(i) ne firstRecChar(i) then do; /*differ*/
         %ResolveVar(CorN=C,index=i, resOut=resultC, valOut= outRecCharVal);
         &sp.Name=varCharName(i);
         &sp.firstRecCharVal= firstRecChar(i);
         &sp.curRecCharVal= curRecChar(i);
         if resultC eq 0 then do;
           numPending+1; /*difference not resolved*/
           outRecChar(i)=""; /*the default resolution*/
         end;
         else  outRecChar(i) = outRecCharVal;
         output &differencesDs;
       end; /*differ*/
     end; /*end of char part*/
```

```
    /*     Numeric Variables     Numeric Variables   ============== */
    do i=1 to &&&sp.cntNum;
      &sp.outRecNumVal=.; /*clear value from prev iteration*/
      if curRecNum(i) ne firstRecNum(i) then do; /*differ*/
             %ResolveVar(CorN=N, index=i, resOut=resultN, valOut=&sp.outRecNumVal);
        &sp.Name=varNumName(i);
        &sp.firstRecCharVal= left(put( firstRecNum(i),  BEST12.));
        &sp.curRecCharVal= left(put( curRecNum(i), BEST12.));
        if resultN eq 0 then do;
          numPending+1; /*difference not resolved*/
          outRecNum(i)=.; /*the default resolution*/
        end;
        else  outRecNum(i) = &sp.outRecNumVal;
        output &differencesDs;
      end; /*differ*/
    end; /* of numeric loop*/

  end; /* end of Not first.&PrimaryKey*/
  if last.&PrimaryKey then do;
    if numPending eq 0 then output &FixedDs;
    else output &FailedDs; /* Partially Fixed */
  end;
  else if not first.&PrimaryKey then output &DeferredDs; /*e.g. &sp.recNum=2*/
RUN;

%* Restore original variable order;
DATA &FixedDs;
  retain &&&sp.origSeqVars;
  set &FixedDs;
RUN;

DATA &DeferredDs;
  retain &&&sp.origSeqVars;
  set &DeferredDs;
RUN;

DATA &FailedDs;
  retain &&&sp.origSeqVars;
  set &FailedDs;
RUN;

%goon: %put >>> all done;
%if &cleanUp eq 1 %then %do;
  PROC  datasets library=work nolist;
    delete &sp.VarMapDs &sp.VarListDs &sp.RuleSpecsDs;
  RUN; QUIT;
%end;
%MEND DiffResolve;

/*Helper MACRO invoked by DiffResolve, internal use only*/
%MACRO ResolveVar(CorN=, index=, resOut=, valOut=);
&resOut = 0;
%let dsid=%sysfunc(open(&sp.RuleSpecsDs));
%syscall set(dsid);
%do %while(%sysfunc(fetch(&dsid)) eq 0) ;
  %if &type = &CorN %then %do;
    if &varIndex eq &index or "&varIndex" eq '&index' then do;
      if %unquote(&pattern) then do;
        &resOut=1;
        &valOut=%unquote(&assign);
        goto &CorN.done;
      end;
    end;
```

```
  %end;
%end;
&CorN.done: /*RULE applied so all DONE*/;
%let rc=%sysfunc(close(&dsid));
%if &rc ne 0 %then %put %sysfunc(sysmsg());
%MEND;
```