Paper 070-2007

Using the DATASETS Procedure Part II

David Fickbohm Golden Gate University. San Francisco CA

ABSTRACT

The purpose of this paper is to continue to explain when, why, and how to use the DATASETS procedure. The DATASETS procedure is a very powerful procedure.

The paper will briefly cover the basic concepts and syntax of the DATASETS procedure.
This paper will discuss in depth:

AGE, APPEND, CHANGE, COPY, DELETE, EXCHANGE, and SAVE. The instructions that manipulate data files;

MODIFY the instruction that can change the attributes of SAS files and, through the use of subordinate statements, the variables in SAS data files.

LABEL the instruction that can assign, change, and remove variable labels for SAS data files specified in the modify statement.

RENAME the instruction that can rename variables for SAS data files specified in the modify statement.

FORMAT and INFORMAT the instructions that can permanently assign, change, and remove variable formats and informats for SAS data files specified in the MODIFY statement.

INTEGRITY CONSTRAINTS will be discussed in depth, the concept, types of, creation of, use of, and deletion of Integrity constraints.

INDEXES will be discussed in depth, the concept, types of, creation of, use of, and deletion of indexes.

AUDIT the concept, creation of, use of, and deletion of audit files will be discussed

CONTENTS the instruction that displays information about SAS data files.

INTRODUCTION

PROC DATASETS is a single PROC containing functionality that is also available in PROCs COPY, CONTENTS, and APPEND. The DATASETS procedure is more efficient, more versatile, and has features far beyond COPY, CONTENTS, and APPEND. Using the DATASETS procedure lets me remember, become proficient, and rely on one procedure rather than waste brain storage and time working with three. This paper will try to explain why, when, and how to use the DATASETS procedure.

BASIC IDEA

A SAS data file contains descriptor information and data values. SAS views hold only descriptors. All of these can be modified by PROC DATASETS.

A SAS library is a collection of one or more SAS files. These files can be called members. Members or files, they are recognized by the SAS System and are referenced and stored as a unit. Each library is associated with a SAS library engine; this is the software interface between SAS and a library.

PROC DATASETS reads and changes information about SAS data files without actually changing the data. This makes PROC DATASETS much more efficient than the data step which must read each record in, whether or not it will be changed, in order to make changes to the descriptor information.

GETTING TO KNOW PROC DATASETS

SYNTAX

The syntax is basically the same for all functions, although defaults do vary, and not all options apply to all statements.

```
PROC DATASETS <options>;
<STATEMENTS> <options>;
QUIT;
```

Options that apply to most statements follow:
DETAILS/NODETAILS – controls the output of observations, variables, and labels
LIST/NOLIST – controls listing of directory members. The default is LIST.

LIBRARY (or LIB) – Specifies the SAS Library. If a library is not specified, the default is the WORK library.
FORCE – Used to force a RUN group to continue to execute even though an error has occurred when processing the RUN group. This also has another implication if you are running the APPEND statement in your DATASETS processing, as it sets the FORCE option on the APPEND..
MEMTYPE – specifies the member type.
NOWARN - suppresses the error processing that occurs when a SAS file that is specified in a SAVE, CHANGE, EXCHANGE, REPAIR, DELETE, or COPY statement or listed as the first SAS file in an AGE statement is not in the input library specified. When an error occurs and the NOWARN option is in effect, PROC DATASETS continues processing that RUN group. If NOWARN is not in effect, PROC DATASETS stops processing that RUN group and issues a warning for all operations except DELETE, for which it does not stop processing.

OPERATION
Being a procedure used to manage data, PROC DATASETS is not intended to create output.

EXECUTION OF STATEMENTS
Statements execute in the order they are written. To see the contents of a data file, copy a data file, and then visually compare the contents of the second data file with the first, the statements that perform those tasks must appear in that order (that is, CONTENTS, COPY, CONTENTS).

It is a very good idea to always include a LIB statement specifying the library which will be worked with.

```
LIBNAME SS06 'C:\SALESJUN05';
PROC DATASETS LIB = SS06; QUIT;
```

RUN-GROUP PROCESSING
PROC DATASETS supports run-group processing. Run-group processing provides the ability to submit run groups without ending the procedure.

The DATASETS procedure supports four types of run groups. Each run group is defined by the statements that compose it and by what causes it to execute.

Some statements in PROC DATASETS act as implied run statements because they cause the run group preceding them to execute.

The following explains what statements compose a run group and what causes each run group to execute:
The PROC DATASETS statement always executes immediately. No other statement is necessary to cause the PROC DATASETS statement to execute.
The MODIFY statement, and any of its subordinate statements, form a run group. The run groups always execute immediately. No other statement is necessary to cause a MODIFY run group to execute.
The APPEND, CONTENTS, and COPY statements (including EXCLUDE and SELECT, if present), form their own run groups. Every APPEND statement forms a single-statement run group; every CONTENTS statement forms a single-statement run group; and every COPY step forms a run group. Any other statements in the procedure, except those that are subordinate to either the COPY or MODIFY statement, cause the run group to execute.
One or more of the following statements form a run group: AGE, CHANGE, DELETE, EXCHANGE, REPAIR, SAVE.

If any of these statements appear in sequence in the PROC step, the sequence forms a run group. To execute the run group, submit one of the following statements: PROC DATASETS, APPEND, CONTENTS, COPY, MODIFY, QUIT, RUN, another DATA or PROC step.

SAS reads the program statements that are associated with one task until it reaches a run statement or an implied run statement. It executes all of the preceding statements immediately, then continues reading until it reaches another run statement or implied run statement. To execute the last task, a run statement or a statement that stops the procedure must be used.

ERROR HANDLING
Generally, if an error occurs in a statement, the run group containing the error does not execute. Run groups preceding or following the one containing the error execute normally. The MODIFY run group is an exception. If a syntax error occurs in a statement subordinate to the MODIFY statement, only the statement containing the error fails. The other statements in the run group execute.

FORCING A RUN GROUP WITH ERROR TO EXECUTE

The FORCE option in the PROC DATASETS statement forces execution of the run group even if one or more statements contain errors. Only the statements that are error-free execute.

ENDING THE PROCEDURE

To stop the DATASETS procedure, a QUIT statement, a RUN CANCEL statement, a new PROC statement, or a DATA statement must be issued. Submitting a QUIT statement executes any statements that have not executed. Submitting a RUN CANCEL statement cancels any statements that have not executed.

CONTENTS STATEMENT IN PROC DATASETS

DETERMINING THE CONTENTS OF A LIBRARY AND THE ATTRIBUTES OF THE VARIABLES IN THE DATASETS.

Using PROC DATASETS here is more efficient than PROC CONTENTS. It requires less keying. PROC DATASETS can produce a report that provides information about all SAS data files that have the type or types specified by the MEMTYPE option. Libref refers to the SAS data library. The default libref is the libref of the procedure input library. Read access to all read-protected SAS data files is required. Information about library members is displayed in the log. The information about the contents of library members is displayed in output.

```
PROC DATASETS LIB = SS06;
    CONTENTS DATA = _ALL_; QUIT;
```

The CONTENTS statement has two very useful options:
The OUT= option lets you write the results of the CONTENTS statement to a separate output data file. The OUT2= option lets you write information about the data file's indexes and integrity constraints to a separate output data file.

```
PROC DATASETS LIB = SS06;
    CONTENTS DATA = DETAIL OUT = SS06.RESULTS;
QUIT;

PROC DATASETS LIB = SS06;
    CONTENTS DATA = MASTR OUT2 = SS06.INDEX_RESULTS;
QUIT;

PROC DATASETS LIB = SS06;
    CONTENTS DATA = MASTR OUT2 = SS06.INT_CONS_RESULTS;
QUIT;
```

COPY AND MOVE STATEMENTS IN PROC DATASETS

COPYING AND MOVING SAS DATASETS BETWEEN LIBRARIES

PROC DATASETS makes copying and moving data files from one library to another very easy. The SELECT and EXCLUDE options can be used to specify or limit which data files within a library are copied from one library to another. Changing the keyword COPY to MOVE moves a member instead of copying it. The member is in both libraries after copying. The member is only in the OUT= library after moving.

```
LIBNAME SS06 'C:\TEMP1';
LIBNAME KEPR 'T:\DAVEKEEP';
PROC DATASETS;
    COPY IN = SS06 OUT = KEPR; QUIT;
```

This copies all SAS members in SS06 to KEPR. If I add a SELECT or EXCLUDE statement I can be more specific. The code below copies only members AAA and BBB to KEPR.

```
LIBNAME SS06 'C:\TEMP1';
LIBNAME KEPR 'T:\DAVEKEEP';
PROC DATASETS;
    COPY IN = SS06 OUT = KEPR;
    SELECT AAA BBB;
```

```
QUIT;
```

The code below copies all members EXCEPT CCC.

```
LIBNAME SS06 'C:\TEMP1';
LIBNAME KEPR 'T:\DAVEKEEP';
PROC DATASETS;
    COPY IN = SS06 OUT = KEPR;
    EXCLUDE CCC;
QUIT;
```

EXCHANGE

This instruction exchanges the names of two SAS files in a SAS library.

```
PROC DATASETS LIB = MYLIB;
   EXCHANGE SALES = JUN_SALES;
QUIT;
```

SAVE OR DELETE A LIBRARY MEMBER, KILL ALL LIBRARY MEMBERS

The SAVE statement specifies which members will be kept in a SAS library. All others, not specified, will be deleted. The DELETE statement specifies which members will be deleted from a SAS library. The KILL statement instructs SAS to delete all members in a library.

The code below will cause all members except AAA and BBB to be deleted.

```
LIBNAME SS06 'C:\TEMP1';
PROC DATASETS LIBRARY = SS06;
    SAVE AAA BBB; QUIT;
```

The code below will only delete library members AAA and BBB.

```
LIBNAME SS06 'C:\TEMP1';
PROC DATASETS LIBRARY = SS06;
    DELETE AAA BBB; QUIT;
```

The code below deletes all members in a library. BE VERY CAREFUL with KILL.

```
LIBNAME SS06 'C:\TEMP1';
PROC DATASETS LIBRARY = SS06 KILL;
QUIT;
```

APPEND STATEMENT IN PROC DATASETS

The APPEND statement adds observations from one data file to another. Unlike the SET statement which reads in all observations from the data files being concatenated, the APPEND statement only reads in observations from the data file being appended. This saves processing time.

The code below appends JUNE05_CLAIMS to MAY05_CLAIMS.

If the two data files being concatenated contain different variables, data types, and/or lengths, the FORCE option can be used. This is a case where the FORCE option is very appropriate and very powerful.

```
PROC DATASETS LIBRARY = WORK FORCE;
    APPEND OUT = MAY05_CLAIMS   DATA = JUNE05_CLAIMS;
QUIT;
```

CHANGE STATEMENT IN PROC DATASETS

The CHANGE statement renames one or more members within a SAS library.
The old member name is specified on the left and the new member name on the right.

```
PROC DATASETS LIBRARY = YY;
    CHANGE MAY05_CLAIMS = MAY_JUNE05_CLAIMS;
```

```
QUIT;
```

AGE STATEMENT IN PROC DATASETS

The AGE statement renames a group of related SAS members in a library.
The code below will add day1 to the library and delete day7.

```
PROC DATASETS LIBRARY=DAILY NOLIST;
    AGE TODAY DAY1-DAY7;
QUIT;
```

```
LOG LISTING
NOTE: Deleting DAILY.DAY7 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY6 to DAILY.DAY7 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY5 to DAILY.DAY6 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY4 to DAILY.DAY5 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY3 to DAILY.DAY4 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY2 to DAILY.DAY3 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY1 to DAILY.DAY2 (memtype=DATA).
NOTE: Ageing the name DAILY.TODAY to DAILY.DAY1 (memtype=DATA).
```

I used the member names day1-day7 for clarity. The member names do not have to describe the data. .

MODIFY STATEMENT IN PROC DATASETS

The MODIFY statement gives the ability to change attributes of a specific library member or attributes of the library member's variables.

The code below works with the member JUN05_PAYMENTS in the work library. It adds a label to the member, it renames a variable, assigns an appropriate label to the newly renamed variable, and finally a format is assigned to an existing variable. The LABEL, RENAME, FORMAT, and, INFORMAT statements can ONLY be used after a modify statement.

```
PROC DATASETS LIBRARY = WORK;
MODIFY JUN05_PAYMENTS (LABEL = 'NEW_MEMBER_LABEL');
  RENAME OLD_VARIABLE_NAME = NEW_VARIABLE_NAME;
  LABEL NEW_VARIABLE_NAME = LABEL_FOR_RENAMED_VARIABLE;
  FORMAT EXISTING_VARIABLE_NAME COMMA11.2;
QUIT;
```

The INFORMAT statement can change or delete an informat.
To change an informat, specify the variables followed by the new informats. To delete an existing informat list the variable without the existing informat. The code below changes the informat for xx and aa to a numeric 2, it changes the variables day1 thru day3 to numeric 4.1 and removes the informat from daily_sales.

```
PROC DATASETS LIBRARY = WORK;
MODIFY AUG07_SALES;
  INFORMAT EXISTING_VARIABLE_NAMES INFORMATS;
  INFORMAT XX AA 2. DAY1-DAY3 4.1 DAILY_SALES;
QUIT;
```

As no observations are read in or written out during processing, for an existing library member, the MODIFY statement is the best way to add a label, rename a variable, or change a format or informat.

AUDIT STATEMENT IN PROC DATASETS

The AUDIT statement takes on one of two forms, depending on whether the audit trail is being initiated, suspended, resumed or terminated.

INITIATE

This creates an audit file that has the same name as the SAS data file and a data file type of AUDIT. The audit file logs additions, deletions, and updates to the SAS data file.

```
PROC DATASETS LIB = MYLIB;
   AUDIT MAY_SALES;
   INITIATE;
QUIT;
```

SUSPEND

This suspends event logging to the audit file, but does NOT delete the audit file.

```
PROC DATASETS LIB = MYLIB;
   AUDIT MAY_SALES;
   SUSPEND;
QUIT;
```

RESUME

This resumes event logging to the audit file, if it was suspended.

```
PROC DATASETS LIB = MYLIB;
   AUDIT MAY_SALES;
   RESUME;
QUIT;
```

TERMINATE

This terminates event logging and deletes the audit file.

```
PROC DATASETS LIB = MYLIB;
   AUDIT MAY_SALES;
   TERMINATE;
QUIT;
```

INTEGRITY CONSTRAINT STATEMENTS IN PROC DATASETS

Integrity constraints are a set of data validation rules that can be specified to restrict the data values accepted into a SAS data file. Using integrity constraints can preserve the correctness and consistency of stored data. SAS enforces the integrity constraints each time data is changed or deleted in a variable that has integrity constraints assigned to it.

There are two kinds of integrity constraints.

- General constraints restrict the data values that are accepted for the variables in a single data file.
  Check – Limits the values of variables to a specific set, range, or list of values.
  Unique/ Distinct - The values of the variables must be unique.
  Not Null - The variable cannot contain a SAS missing value.

  Primary Key – specifies a primary key, that is, a set of variables that does not contain missing values and whose values are unique.

- Referential constraints provide the ability to link the data values of specific variables in one data file to specific variables in another data file. A referential integrity constraint is created when a primary key integrity constraint in one data file is referenced by a foreign key integrity constraint in another data file.
- Foreign Key – specifies a foreign key, that is, a set of variables whose values are linked to the values of the primary key variables in another data file. The referential actions are enforced when updates are made to the values of a primary key variable that is referenced by a foreign key.
  - RESTRICT - prevents the data values in the primary key from being updated or deleted unless there are no matching data values in any referencing foreign key data files. This is the default if no action is specified.
  - NULL - allows primary key variables to be updated or deleted, but changes any matching data values in the foreign key data files to null.
  - CASCADE – allows primary key variables to be updated AND modifies matching data values in the foreign key data files to the same value. CASCADE is not supported for delete operations.

IC CREATE

This instruction creates an integrity constraint.

The syntax is:

IC CREATE <Constraint name> CONSTRAINT <Message = 'message string' <MSGTYPE=USER>>;

<Constraint name> is an optional name for the constraint. The name must be a valid SAS name. It is good practice to assign a constraint name.

CONSTRAINT is a required argument. The syntax is:
NOT NULL(VARIABLE)
UNIQUE(VARIABLES)
DISTINCT(VARIABLES)
CHECK(WHERE = expression)
PRIMARY KEY(VARIABLES)
FOREIGN KEY (VARIABLES) REFERENCES table name <ON DELETE referential action>
<ON UPDATE referential action>

<MESSAGE = 'message string' is the text of a user defined error message written to the log when the data fails the constraint.
<MSGTYPE = USER>> controls the format of the integrity constraint message. If the message = option is used, a message is inserted into the SAS error message, separated by a space. MSGTYPE = USER suppresses the SAS portion of the message.

Examples

```
PROC DATASETS NOLIST;
 MODIFY T_SALES;
    IC CREATE SSN_NN = not null(SSN) message = 'invalid ssn';
    IC CREATE SSN_UNIQUE = unique(SSN) ;
    IC CREATE SSN_PROPID_DISTINCT = distinct(SSN PROPID);
    IC CREATE HOMECOST_VALUE = check(where= (homecost  > 0 or VALUE  = 99));
QUIT;

PROC SORT DATA = ALL_PROFILES01;  BY PROPID; RUN;
PROC DATASETS NOLIST LIB = WORK;
    MODIFY ALL_PROFILES01;
    IC CREATE PRMKEY = primary key (prodid);
QUIT;

PROC SORT DATA = T_PROPERTY;  BY PROPID;  RUN;
PROC DATASETS NOLIST LIB = WORK;
    MODIFY T_PROPERTY;
       IC CREATE FORKEY  = foreign key (propid) REFERENCES ALL_PROFILES01
       On update cascade on delete set null;
QUIT;
```

IC DELETE

This instruction deletes a constraint or constraints.
The syntax is:

```
IC DELETE constraint-name(s) | _ALL_ ;
```

Examples

```
PROC DATASETS NOLIST;
 MODFIY T_SALES;
     IC DELETE SSN_NN SSN_UNIQUE;
     IC DELETE  _ALL_;
```

would delete the SSN_NN and SSN_UNIQUE constraints.
would delete all constraints for the SAS data file specified in the preceding MODIFY statement.

Copying entire libraries can result in data files residing in the output data library that contain inactive foreign key constraints. This occurs when a foreign key exists in a data file within the library being copied and it references a primary key residing in a different library that was not involved in the library copy. The foreign key residing in the output library is in an inactive state and must be reactivated before updates can be made to the output data file. The foreign key in the original input data file is still active. The IC REACTIVATE statement is used to reactivate inactive foreign key constraints.

IC REACTIVATE

This instruction reactivates a foreign key integrity constraint that is inactive.
The syntax is:

```
IC REACTIVATE foreign-key-name REFERENCES libref;
```

Foreign-key-name is the name of the foreign key to reactivate.
Libref refers to the SAS library containing the data file that holds the primary key that is referenced by the foreign key.

Examples

```
PROC DATASETS LIBRARY = MYLIB;
    MODIFY MYOWN;
       IC REACTIVATE FORKEY REFERENCES MAINLIB;
QUIT;
```

INDEX STATEMENTS IN PROC DATASETS

SAS indexes can dramatically improve the performance of programs that access small subsets of observations from large SAS data files. Searching through SAS online documentation for information on indexes using a search feature is very much like using searching an indexed file. Searching a non indexed file is like searching going through online documentation page by page by page.

An index is a SAS file separate from the data file, though logically related to the data file. The index file is stored in the same SAS data library as the SAS data file. The index file is automatically associated with the data file when is it created. More than one index can exist on a single data file, and all indexes associated with the given data file are stored in a single index file. Changes to a data file cause the index file to be automatically updated. The CONTENTS statement in PROC DATASETS can be used to display information on indexes. To display information regarding the use of indexes in the log, use the system option **OPTION MSGLEVEL = I;**

Types of indexes

There are two types of indexes: simple and complex. A simple index has only one key variable. A key variable is the variable upon which the data file is indexed. Multiple simple indexes can be associated with a given data file. The name of the simple index is the same as the key variable. A complex index is one in which the values of two or more variables have been concatenated together to form one value. When creating a complex index, the complex index must be given a name.

Options

There are two options when creating indexes: UNIQUE and NOMISS. The UNIQUE option declares that the values for a variable are unique. An employee_id would be an example. If an attempt is made to add an observation with a duplicate value, that update will be rejected. The NOMISS option instructs SAS to **not** include missing values in the index. The can make the index more efficient if there are many missing data points in the key variable. Observations with missing values on the key variable can be added to a data file where the NOMISS option is in effect.

Centiles

The word 'centile' is a contraction of the words "cumulative percentiles". Centiles are a built-in feature of SAS indexes. SAS uses centiles to determine if it is efficient to use an index to read a SAS data file. SAS initializes each index's centiles when the index is created and keeps them updated when a certain percentage of the index key variable's have been changed within the SAS data file. The default is 5%. This means that when 5% of the key index variables in the SAS data file have their values changed, SAS will recompute the centiles for the index.

Creating Indexes

On the PROC DATASETS statement you may specify the input library using the library option. On the MODIFY statement the name of the data file where the index will be created must be specified. On the INDEX statement the key variable or name and the wanted options are specified. The UPDATECENTILES option specifies when centiles will be updated. Specifying a value of ALWAYS for the UPDATECENTILES option indicates that centiles are updated each time the SAS data file is closed. Specifying a value of NEVER for the UPDATECENTILES option specifies that centiles will not be updated. A percentage other than 5% can be specified.

```
PROC DATASETS LIBRARY = LIBREF;
    MODIFY SAS-DATA-SET;
       INDEX CREATE VARLIST / UNIQUE NOMISS UPDATECENTILES = ALWAYS | NEVER | INTEGER;
QUIT;
```

The example below creates a simple index.

```
LIBNAME TESTLIB 'D:\DAVETEST';
PROC DATASETS LIBRARY = TESTLIB;
    MODIFY ABC123;
```

```
       INDEX CREATE STUDENT_ID / UNIQUE UPDATECENTILES = ALWAYS;
QUIT;
```

The example below creates a complex index.

```
LIBNAME TESTLIB 'D:\DAVETEST';
PROC DATASETS LIBRARY = TESTLIB;
    MODIFY ABC123;
        INDEX CREATE DEM = (GENDER RACE) / UPDATECENTILES = 10;
QUIT;
```

Deleting Indexes

One, multiple, or all indexes associated with a SAS data file may be deleted with the INDEX DELETE statement.

The example below deletes one index associated with dataset abc123. This example deletes a complex index so the name of the index is used.

```
LIBNAME TESTLIB 'D:\DAVETEST';
PROC DATASETS LIBRARY = TESTLIB;
    MODIFY ABC123;
       INDEX DELETE DEM;
QUIT;
```

The example below deletes multiple indexes associated with dataset abc123. This example deletes a complex and a simple index, so the name of the complex index and the name of the variable the simple index is based on are used.

```
LIBNAME TESTLIB 'D:\DAVETEST';
PROC DATASETS LIBRARY = TESTLIB;
    MODIFY ABC123;
      INDEX DELETE DEM STUDENT_ID;
QUIT;
```

The example below deletes all indexes associated with dataset abc123.

```
LIBNAME TESTLIB 'D:\DAVETEST';
PROC DATASETS LIBRARY = TESTLIB;
    MODIFY ABC123;
       INDEX DELETE _ALL_ ;
QUIT;
```

Manipulating Centiles

The INDEX CENTILES statement provides the ability to update the centile information for indexed variables.

The syntax for the INDEX CENTILES statement is:

INDEX CENTILES index(s);
The options are REFRESH which updates the centiles immediately and UPDATECENTILES. The allowable values for the UPDATECENTILES option are explained above.

The example below updates the centiles immediately. This is a complex index so an index name is used.

```
LIBNAME TESTLIB 'D:\DAVETEST';
PROC DATASETS LIBRARY = TESTLIB;
    MODIFY ABC123;
       INDEX CENTILES DEM / REFRESH ;
QUIT;
```

The example below changes the centile percentage from 10% to 20%. This is a simple index so the variable name the data file is indexed on is used.

```
LIBNAME TESTLIB 'D:\DAVETEST';
PROC DATASETS LIBRARY = TESTLIB;
    MODIFY ABC123;
       INDEX CENTILES  PROPID / UPDATECENTILES = 20;
QUIT;
```

REPAIR STATEMENT IN PROC DATASETS

Repair attempts to restore damaged data files or catalogs to a usable condition.
Situations where REPAIR could be useful:

- A system failure occurs while a SAS data file or catalog is being updated
- The device on which a SAS data file or an associated index resides is damaged.
- The disk that stores the SAS data or catalog becomes full before the file is completely written to disk.
- An I/O error occurs while a SAS data file or catalog is being written.

The syntax for the REPAIR statement is:

```
REPAIR SAS-File(s)
```

The following example repairs the junesales data file:

```
PROC DATASETS LIB = WORK;
    REPAIR JUNESALES;
QUIT;
```

CONCLUSION

I believe I have shown that PROC DATASETS is a many featured, multi-faceted, very useful tool. Since PROC DATASETS does not read in nor write out observations, PROC DATASETS is very efficient. It is simple to use and easy to learn.

Caution should be exercised when working with PROC DATASETS. Realize that PROC DATASETS is an interactive procedure, so all statements execute immediately and in the order they appear.

REFERENCES

SAS Institute Inc. (2005) SAS OnlineDoc@ 9.1.3 Cary NC. SAS Institute Inc.

ACKNOWLEDGMENTS

I would like to thank Sunil Gupta, the Coder's Corner Section Chair, for accepting my paper. I would like to thank the leadership of SAS Global Forum for a wonderful conference. I would like to thank Claire Bonney, SAS Tech Support for her patience and assistance. This paper is a far better product because of the patience and encouragement of Diane Olson, SAS I/O Development, and Gary Franklin, SAS Base Engine Development. Finally this paper would not have happened without the understanding and support of my wife Nancy. I humbly thank everyone.

CONTACT INFORMATION

Please feel free to contact me with questions and comments about this paper:

David Fickbohm
Golden Gate University
536 Mission Street
San Francisco, CA 94105
510 594 4151 day phone
510 655 0848 fax
dfickbohm@ggu.edu