SESUG Paper 133-2023

Modernizing Legacy SAS® Applications and Program Code

Kirk Paul Lafler, sasNerd
Clark Roberts, Decision Analytics

Abstract

Whether you are a novice or experienced SAS® programmer with responsibility for the support of your organization's legacy applications, programs and code, assistance is available to help modernize and streamline code for the 21st century and beyond. This paper explores the available constructs, statements, functions, algorithms, operators, methods, expressions, programming techniques and approaches users have to update and modernize legacy applications, programs and code first introduced as far back as the 1970's. Users are provided techniques to streamline, scale and modernize code constructs, statements, functions, algorithms, and legacy applications and program code, and an introduction to the SCAPROC procedure – the SAS code analyzer – to analyze metadata about the contents of SAS code.

Table of Contents

Introduction

As SAS users around the world celebrate a milestone of more than 40-years using SAS software, organizations want, and need, to look at ways to modernize their inventories of legacy SAS applications and program code to operate in the 21st century and beyond. This means that IT personnel, systems and data analysts, SAS® programmers, end-users, management, and stakeholders everywhere must assume the responsibility of identifying, and modernizing, mission-critical legacy applications and program code, with newer, and more scalable and efficient, statements, functions, options, code constructs, algorithms, and programming techniques. This paper explores many areas for consideration, and provides tips, techniques and examples to help guide SAS users through the modernization process. We suggest and illustrate a foundation of technologies, techniques, and approaches to consider while modernizing applications and program code. We also provide an example on the use of the SCAPROC procedure – the SAS code analyzer – to analyze metadata about the contents of SAS code, and to streamline, scale and modernize code constructs, algorithms, functions, and legacy application program code.

What are Legacy Applications and Program Code?

Programmers and application developers have different interpretations and meanings for what legacy code means. For some legacy applications and program code refers to code that someone else wrote a long time ago and as a result may not utilize or represent the latest technologies. Complicating matters further, often, the original developer(s) and/or programmer(s) is/are no longer available or affiliated with the organization. For others, legacy code represents a foundation of older, and often outdated, statements, functions, options, coding constructs, algorithms and other techniques causing applications and program code to become woefully behind 21st century standards.

Other characteristics attributed to legacy code are the unlimited number of coding styles and modifications that occurs over time. This often translates into a legacy code base that may have been originally well-written but evolves into a complex, and less than maintainable, maze of spaghetti code. In these situations the legacy code base is no longer engineered but begins to take on the characteristics of a patched and tangled control structure. Making matters worse, this maze of patched, complex and confusing legacy code often survives without the existence of up-to-date and effective program documentation.

Signs that a Legacy Application May Need Modernizing

Green (2017) describes modernization as, "***. . . newer, faster, sleeker and more useful – all qualities we want our applications to have.*** " Modernizing legacy applications and program code involves an incremental and structured approach. It consists of identifying the target application and program code; selecting potential solutions to use; and finally, implementing structured and scalable solutions to replace varied coding styles and conventions made over its functional life. So, how does an organization know when an application needs modernizing? Green shares five signs to answer this question.

1. Operation and maintenance costs are high.
2. It's clunky or uses outdated technologies.
3. Your business processes have changed.
4. There's no tight integration with future applications.
5. It's not mobile-ready.

So, how should an organization proceed with the modernization of a legacy application and program code project? The best place to start is to get all stakeholders on-board and in agreement with the objectives and changes to be made. Next, obtain the necessary funding for performing the project work, modernizing a legacy application conjures concern from everyone involved. All too often, stakeholders develop a, "If it's not broke then leave it alone!" attitude. To help alleviate the issues associated with modernizing legacy applications and program code, we recommend a five-step modernization approach.

1. Identify mission-critical applications and program code that is/are indispensable to the organization.

2. Review and understand the code associated with the user-interface, the data sources being accessed, the processing requirements, and finally the output and results.

3. Identify and modernize older technologies; hard to modify and inflexible code; and inefficient statements, functions, options and their settings, code constructs, algorithms, and programming techniques with newer and more efficient methods and techniques.

4. Test, Train and Deploy the modernized application and program code to bring all stakeholders on board.

5. Maintain and Support the modernized applications and program code to ensure their flexibility and adaptability to changing requirements, environments and technologies.

Conditional Logic Scenarios

A powerful and necessary programming technique in the SAS® software is its ability to perform different actions depending on whether a programmer-specified condition evaluates to true or false. The method for accomplishing this is to use one or more conditional statements, expressions, and constructs to build a level of intelligence in a program or application. Conditional logic scenarios in the DATA step are frequently implemented using IF-THEN / ELSE and SELECT statements. The SQL procedure also supports logic scenarios and is implemented with a coding technique known as a CASE expression.

Conditional Logic with IF-THEN / ELSE

The IF-THEN / ELSE construct in the DATA step enables a sequence of conditions to be assigned that when executed proceeds through the sequence of logic conditions until a match in an expression is found or until all conditions are exhausted. The example shows a character variable Movie_Length being assigned a value of either “Shorter Length”, “Average Length”, or “Longer Length” based on the mutually exclusive conditions specified in the IF-THEN and ELSE conditions. Although not required, an ELSE condition serves as an effective technique for continuing processing to the next specified condition when a match is not found. An ELSE condition can also be useful as a “catch-all” to prevent a missing value from being assigned.

IF-THEN / ELSE Code:

```
LIBNAME MYDATA “E:/WORKSHOPS/WORKSHOP DATA” ;
DATA IF_THEN_EXAMPLE ;
  ATTRIB Movie_Length LENGTH=$14 LABEL=’Movie Length’ ;
  SET MYDATA.MOVIES ;
  IF LENGTH < 120 THEN Movie_Length = ‘Shorter Length’ ;
  ELSE IF LENGTH > 160 THEN Movie_Length = ‘Longer Length’ ;
  ELSE Movie_Length = ‘Average Length’ ;
RUN ;
PROC PRINT DATA=IF_THEN_EXAMPLE NOOBS ;
  VAR TITLE LENGTH Movie_Length ;
RUN ;
```

IF-THEN / ELSE Results:

Title	Length	Movie_Length
Brave Heart	177	Longer Length
Casablanca	103	Shorter Length
Christmas Vacation	97	Shorter Length
Coming to America	116	Shorter Length
Dracula	130	Average Length
Dressed to Kill	105	Shorter Length
Forrest Gump	142	Average Length
Ghost	127	Average Length
Jaws	125	Average Length
Jurassic Park	127	Average Length
Lethal Weapon	110	Shorter Length
Michael	106	Shorter Length
National Lampoon's Vacation	98	Shorter Length
Poltergeist	115	Shorter Length
Rocky	120	Average Length
Scarface	170	Longer Length
Silence of the Lambs	118	Shorter Length
Star Wars	124	Average Length
The Hunt for Red October	135	Average Length
The Terminator	108	Shorter Length
The Wizard of Oz	101	Shorter Length
Titanic	194	Longer Length

Conditional Logic with SELECT-WHEN / OTHERWISE

Another form of conditional logic available to users is a **SELECT** statement. Its purpose is to enable a sequence of logic conditions to be constructed in a DATA step by specifying one or more **WHEN** conditions and an optional **OTHERWISE** condition. When executed, processing continues through each WHEN condition until a match is found that satisfies the specified expression. Typically one or more WHEN conditions are specified in descending frequency order representing a series of conditions. The next example shows a value based on the mutually exclusive conditions specified in the sequence of logic conditions of “Shorter Length”, “Average Length”, or “Longer Length” being assigned to the character variable Movie_Length. Although not required, the OTHERWISE condition can be useful in the assignment of a specific value or as a “catch-all” to prevent a missing value from being assigned.

SELECT-WHEN / OTHERWISE Code:

```
LIBNAME MYDATA “E:/WORKSHOPS/WORKSHOP DATA” ;
DATA SELECT_EXAMPLE ;
  SET MYDATA.MOVIES ;
  SELECT ;
    WHEN (LENGTH < 120) Movie_Length = ‘Shorter Length’ ;
    WHEN (LENGTH > 160) Movie_Length = ‘Longer Length’ ;
    OTHERWISE Movie_Length = ‘Average Length’ ;
  END ;
RUN ;

PROC PRINT DATA=SELECT_EXAMPLE NOOBS ;
  VAR TITLE LENGTH Movie_Length ;
RUN ;
```

SELECT-WHEN / OTHERWISE Results:

Title	Length	Movie_Length
Brave Heart	177	Longer Length
Casablanca	103	Shorter Length
Christmas Vacation	97	Shorter Length
Coming to America	116	Shorter Length
Dracula	130	Average Length
Dressed to Kill	105	Shorter Length
Forrest Gump	142	Average Length
Ghost	127	Average Length
Jaws	125	Average Length
Jurassic Park	127	Average Length
Lethal Weapon	110	Shorter Length
Michael	106	Shorter Length
National Lampoon's Vacation	98	Shorter Length
Poltergeist	115	Shorter Length
Rocky	120	Average Length
Scarface	170	Longer Length
Silence of the Lambs	118	Shorter Length
Star Wars	124	Average Length
The Hunt for Red October	135	Average Length
The Terminator	108	Shorter Length
The Wizard of Oz	101	Shorter Length
Titanic	194	Longer Length

Conditional Logic with CASE Expressions

Another form of conditional logic available to users is a case expression. Its purpose is to provide a way of conditionally selecting result values from each row in a table (or view). Similar to an IF-THEN/ELSE or SELECT construct in the DATA step, a case expression can only be specified in the SQL procedure. It supports a WHEN-THEN clause to conditionally process some but not all the rows in a table. An optional ELSE expression can be specified to handle an alternative action should none of the expression(s) identified in the WHEN condition(s) not be satisfied. A case expression must be a valid SQL expression and conform to syntax rules similar to DATA step SELECT-WHEN statements. Even though this topic is best explained by example, a quick look at the syntax follows.

```
CASE <column-name>
    WHEN when-condition THEN result-expression
   <WHEN when-condition THEN result-expression> …

   <ELSE result-expression>
END
```

A column-name can optionally be specified as part of the CASE-expression. If present, it is automatically made available to each when-condition, and is classified as a simple case expression. When it is not specified, the column-name must be coded in each when-condition, and is classified as a searched case expression. If a when-condition is satisfied by a row in a table (or view), then it is considered “true” and the result-expression following the THEN keyword is processed. The remaining WHEN conditions in the case expression are skipped. If a when-condition is “false”, the next when-condition is evaluated. SQL evaluates each when-condition until a “true” condition is found or in the event all when-conditions are “false”, it then executes the ELSE expression and assigns its value to the CASE expression’s result. A missing value is assigned to a case expression when an ELSE expression is not specified and each when-condition is “false”.

In the next example, a **searched case expression** is illustrated. A searched case expression in the SQL procedure provides users with the capability to perform more complex comparisons. Although the number of keystrokes can be more than with a simple case expression, the searched case expression offers the greatest flexibility and is the primary form used by SQL’ers. The

noticeable absence of a column name as part of the case expression permits any number of columns to be specified from the underlying table(s) in the WHEN-THEN/ELSE logic scenarios.

The next example shows a searched case expression being used to assign the character variable Movie_Length with the AS keyword. A value of “Shorter Length” for movie lengths less than 120 minutes, “Longer Length” for movie lengths greater than 160 minutes, or “Average Length” for all other movie lengths is assigned to the newly created column. Although not required, an ELSE condition can be useful in the assignment of a specific value or as a “catch-all” to prevent a missing value from being assigned.

Searched CASE Expression Code:

```
LIBNAME MYDATA “E:/WORKSHOPS/WORKSHOP DATA” ;
PROC SQL;
  SELECT TITLE,
         LENGTH,
         CASE
           WHEN LENGTH < 120 THEN 'Shorter Length'
           WHEN LENGTH > 160 THEN 'Longer Length'
           ELSE 'Average Length'
         END AS Movie_Length
    FROM MYDATA.MOVIES ;
QUIT ;
```

Searched CASE Expression Results:

Title	Length	Movie_Length
Brave Heart	177	Longer Length
Casablanca	103	Shorter Length
Christmas Vacation	97	Shorter Length
Coming to America	116	Shorter Length
Dracula	130	Average Length
Dressed to Kill	105	Shorter Length
Forrest Gump	142	Average Length
Ghost	127	Average Length
Jaws	125	Average Length
Jurassic Park	127	Average Length
Lethal Weapon	110	Shorter Length
Michael	106	Shorter Length
National Lampoon's Vacation	98	Shorter Length
Poltergeist	115	Shorter Length
Rocky	120	Average Length
Scarface	170	Longer Length
Silence of the Lambs	118	Shorter Length
Star Wars	124	Average Length
The Hunt for Red October	135	Average Length
The Terminator	108	Shorter Length
The Wizard of Oz	101	Shorter Length
Titanic	194	Longer Length

As previously mentioned, searched case expressions provide users with the capability to perform more complex logic comparisons. Combined with logical and comparison operators, searched case expressions along with their WHERE clause counterparts, provide the capabilities to construct complex logic scenarios. In the next example a listing of “Action” and “Comedy” movies are displayed. Using a searched case expression, a value of “Shorter Length” for movie lengths less than 120 minutes, “Longer Length” for movie lengths greater than 160 minutes, or “Average Length” for all other movie lengths is assigned to the newly created column. A column heading of Movie_Type is assigned to the new column with the AS keyword.

Searched CASE Expression Code:

```
LIBNAME MYDATA "E:/WORKSHOPS/WORKSHOP DATA" ;
PROC SQL;
  SELECT TITLE, RATING, LENGTH, CATEGORY,
    CASE
      WHEN UPCASE(CATEGORY) CONTAINS 'ACTION' AND LENGTH < 120 THEN 'Action Short'
      WHEN UPCASE(CATEGORY) CONTAINS 'ACTION' AND LENGTH > 160 THEN 'Action Long'
      WHEN UPCASE(CATEGORY) CONTAINS 'ACTION' AND
           LENGTH BETWEEN 120 AND 160 THEN 'Action Medium'
      WHEN UPCASE(CATEGORY) CONTAINS 'COMEDY' AND LENGTH < 120 THEN 'Comedy Short'
      WHEN UPCASE(CATEGORY) CONTAINS 'COMEDY' AND LENGTH > 160 THEN 'Comedy Long'
      WHEN UPCASE(CATEGORY) CONTAINS 'COMEDY' AND
           LENGTH BETWEEN 120 AND 160 THEN 'Comedy Medium'
      ELSE 'Not Interested'
    END AS MOVIE_TYPE
    FROM MYDATA.MOVIES
      WHERE UPCASE(CATEGORY) CONTAINS 'ACTION' OR 'COMEDY';
QUIT;
```

Searched CASE Expression Results:

Title	Rating	Length	Category	MOVIE_TYPE
Brave Heart	R	177	Action Adventure	Action Long
Casablanca	PG	103	Drama	Not Interested
Christmas Vacation	PG-13	97	Comedy	Comedy Short
Coming to America	R	116	Comedy	Comedy Short
Dracula	R	130	Horror	Not Interested
Dressed to Kill	R	105	Drama Mysteries	Not Interested
Forrest Gump	PG-13	142	Drama	Not Interested
Ghost	PG-13	127	Drama Romance	Not Interested
Jaws	PG	125	Action Adventure	Action Medium
Jurassic Park	PG-13	127	Action	Action Medium
Lethal Weapon	R	110	Action Cops & Robber	Action Short
Michael	PG-13	106	Drama	Not Interested
National Lampoon's Vacation	PG-13	98	Comedy	Comedy Short
Poltergeist	PG	115	Horror	Not Interested
Rocky	PG	120	Action Adventure	Action Medium
Scarface	R	170	Action Cops & Robber	Action Long
Silence of the Lambs	R	118	Drama Suspense	Not Interested
Star Wars	PG	124	Action Sci-Fi	Action Medium
The Hunt for Red October	PG	135	Action Adventure	Action Medium
The Terminator	R	108	Action Sci-Fi	Action Short
The Wizard of Oz	G	101	Adventure	Not Interested
Titanic	PG-13	194	Drama Romance	Not Interested

Subsetting with WHERE Expressions in a PROCedure

Gupta (2006) describes using a subsetting-IF versus a WHERE-statement or WHERE= data set option to subset observations. To avoid using a subsetting-IF statement in a DATA step, SAS users may be able to specify a WHERE= data set option for subsetting purposes directly in a procedure. This approach prevents the creation of a data set and, as a result, is more likely to scale better by reducing CPU and I/O resources. Gupta emphasizes an important detail that all SAS users should know when specifying a WHERE condition in a procedure, "Multiple WHERE conditions within SAS procedures are not cumulative as they are in a DATA step meaning the most recent WHERE condition replaces any, and all, previously specified WHERE condition(s)."

PROC PRINT with WHERE Expression Code:

```
/* WHERE Statement to Subset Observations */
proc print data=sashelp.cars noobs ;
  where type="SUV" or type="Wagon" ;
run ;

            < or >

/* WHERE= Data Set Option to Subset Observations */
proc print data=sashelp.cars(where=(type="SUV" or type="Wagon")) noobs ;
run ;
```

Using the IN Operator for Comparisons

Legacy SAS applications and program code often use one, or more, OR comparison operators to handle logic scenarios. Although syntactically correct, a series of individual comparisons separated by an OR comparison operator is generally less efficient than using an IN operator. The reason is due to the way an IN operator operates. When an IN operator is specified, SAS stops making comparisons as soon as it finds a match. This is not the case with an OR operator. In the next example, a number of individual comparisons are specified using an OR operator.

OR Comparison Operator Code:

```
PROC SQL ;
  SELECT Origin, Type, MSRP
   FROM SASHELP.Cars
    WHERE Type = "SUV"
       OR Type = "Truck"
       OR Type = "Wagon"
     ORDER BY MSRP ;
QUIT ;
```

In the next example, an IN operator is specified to help modernize the process of handling a number of individual comparisons. The IN operator provides a convenient, and concise, way to specify scenarios with many OR comparisons. *A similar example using DS2 SQLSTMT package approach can be found here.*

IN Operator Code:

```
PROC SQL ;
  SELECT Origin, Type, MSRP
   FROM SASHELP.Cars
    WHERE Type IN ( "SUV","Truck","Wagon" )
     ORDER BY MSRP ;
QUIT ;
```

Concatenating Strings and Variables with CAT Functions

SAS functions serve an essential role in the Base SAS software. Representing a variety of built-in and callable routines, functions serve as the "work horses" in the SAS software providing users with "ready-to-use" tools designed to ease the burden of writing and testing often lengthy and complex code for a variety of programming tasks. The advantage of using SAS functions is evident by their relative ease of use, and their ability to provide a more efficient, robust and scalable approach to simplifying a process or programming task. In this example, we show how the TRIM and LEFT functions along with the concatenate operator to concatenate strings and variables together can be replaced with the CAT functions.

CAT Function Code:

```
data _null_ ;
  length NUM 3. A B C D E $ 8 BLANK $ 1 ;
  A = 'The' ;
  NUM = 5 ;
  B = ' Cats' ;
  C = 'in' ;
  D = ' the' ;
  E = 'Hat' ;
  BLANK = ' ' ;

 *Old concatenation approach with TRIM and LEFT functions and concatenation
  operator ;
  OLD=trim(left(A)) || BLANK || trim(left(NUM)) || BLANK || trim(left(B)) ||
      BLANK || trim(left(C)) || BLANK || trim(left(D)) || BLANK || trim(left(E)) ;

 * Using the CAT functions to concatenate character and numeric values together ;
❶ CAT  = cat (A, NUM, B, C, D, E) ;
❷ CATQ = catq(BLANK, A, NUM, B, C, D, E) ;
❸ CATS = cats(A, NUM, B, C, D, E) ;
❹ CATT = catt(A, NUM, B, C, D, E) ;
❺ CATX = catx(BLANK, A, NUM, B, C, D, E) ;
   put OLD= / STRIP= / CAT= / CATQ= / CATS= / CATT= / CATX= / ;
run ;
```

CAT Function Results:

```
OLD=The 5 Cats in the Hat
CAT=The     5 Cats   in       the    Hat
CATQ="The     " 5 " Cats   " "in      " " the    " "Hat     "
CATS=The5CatsintheHat
CATT=The5 Catsin theHat
CATX=The 5 Cats in the Hat
```

Analysis:

In the example, above, a single numeric variable, NUM, and six character variables: A, B, C, D, E, and BLANK are defined with their respective values as: NUM=5, A='The', B=' Cats', C='in', D=' the', E='Hat' and BLANK=' '. The oldest way of concatenating two or more strings or variables together is then specified, using the TRIM and LEFT functions with the concatenation operator "||" in an assignment statement. As an alternative, a newer and more robust concatenation approach is specified using the CAT family of functions: CAT, CATQ, CATS, CATT, and CATX.

❶ **CAT**, the simplest of concatenation functions, joins two or more strings and/or variables together, end-to-end producing the same results as with the concatenation (double bar) operator.

❷ **CATQ** is similar to the default features of the CATX function, but the CATQ function adds quotation marks to any concatenated string or variable.

❸ **CATS** removes leading and trailing blanks and concatenates two or more strings and/or variables together.

❹ **CATT** removes trailing blanks and concatenates two or more strings and/or variables together.

❺ **CATX**, perhaps the most robust CAT function, removes leading and trailing blanks and concatenates two or more strings and/or variables together with a user-specified delimiter between each.

Concatenating (or Appending) Data Sets

Concatenating data sets is the process of combining two, or more, data sets, one after the other, with the purpose of creating a single data set. The number of observations in the new data set is the sum total of observations in all the original input data sets. The order of observations in the concatenated data set is arranged sequentially with the observations from the first data set, followed by the observations from the second data set, and so on. The concatenated data set contains the same variables as the input data sets. Should an input data set contain different variables from the other input data sets, the concatenated data set will have missing values assigned to the variables from the other input data sets.

Concatenating with the DATA-SET-RUN Construct

SAS provides users with a few ways to concatenate data sets. In the first example, below, an old-style DATA-SET construct is specified to concatenate the two data sets, RUGs_2015 and RUGs_2016. Although syntactically correct, this approach does not scale well because it forces SAS to incur heavy I/O (input/output) because the observations in each input data set must be read and written to the concatenated data set.

DATA–SET-RUN Code:

```
data Concatenated_Results ;
  set RUGs_2015
      RUGs_2016 ;
run ;
```

DATA–SET-RUN Results:

RUG	Number_Papers	Year
MWSUG	96	2015
SCSUG	29	2015
SESUG	148	2015
WUSS	102	2015
MWSUG	124	2016
SCSUG	62	2016
SESUG	148	2016
WUSS	112	2016

Concatenating with a PROC SQL Outer Union CORR

A second approach uses PROC SQL to concatenate data sets. In this next example, an OUTER UNION CORR set operator is specified, and SQL reads and processes the tables in each query producing a new concatenated table of results.

PROC SQL Code:

```
proc sql ;
 create table Concatenated_Results as
  select * from RUGs_2015
 outer union corr
  select * from RUGs_2016 ;
  select * from Concatenated_Results ;
quit ;
```

PROC SQL Results:

RUG	Number_Papers	Year
MWSUG	96	2015
SCSUG	29	2015
SESUG	148	2015
WUSS	102	2015
MWSUG	124	2016
SCSUG	62	2016
SESUG	148	2016
WUSS	112	2016

Concatenating with PROC APPEND (or PROC DATASETS – APPEND Statement)

A third, and more efficient, concatenation approach is available to SAS users. Using PROC APPEND (or the APPEND statement in PROC DATASETS), an input data set can be appended to another data set. The advantage of using this approach is reduced I/O, since SAS does not have to read the observations in the base data set. Appending this way offers a way to scale an application. As the number of observations in the base data set grows, the advantage of using this approach can become huge. In the next example, two PROC APPENDs are specified to concatenate the observations in the RUGs_2015 and RUGs_2016 data sets.

PROC APPEND Code:

```
proc append base=Concatenated_Results
            Data=RUGs_2015 ;
run ;
proc append base=Concatenated_Results
            Data=RUGs_2016 ;
run ;
```

PROC APPEND Results:

RUG	Number_Papers	Year
MWSUG	96	2015
SCSUG	29	2015
SESUG	148	2015
WUSS	102	2015
MWSUG	124	2016
SCSUG	62	2016
SESUG	148	2016
WUSS	112	2016

Processing Multiple TABLE Statements with PROC FREQ

Benjamin (2012) describes a common problem programmers have when using PROC FREQ to produce multiple table results. Programmers will often code two, or more, individual PROC FREQ and TABLE statements even for the same input data set. Although the PROC FREQ code, illustrated below, is syntactically correct, invoking PROC FREQ multiple times in this way can result in an increase in the amount of time for processing the request.

PROC FREQ Code:

```
proc freq data=sashelp.cars ;
  table Origin / list out=work.Origin_Freq1 ;
run ;
proc freq data=sashelp.cars ;
  table Origin * Type / list out=work.Origin_Freq2 ;
run ;
proc freq data=sashelp.cars ;
  table Origin * Type * Cylinders / list out=work.Origin_Freq3 ;
run ;
```

To optimize the code, programmers can force a single pass over the input data set and as a result reduce the amount of processing time needed to produce the resulting data sets, as follows.

Optimized PROC FREQ Code:

```
proc freq data=sashelp.cars ;
  table Origin / list out=work.Origin_Freq1 ;
  table Origin * Type / list out=work.Origin_Freq2 ;
  table Origin * Type * Cylinders / list out=work.Origin_Freq3 ;
run ;
```

List of Procedures Supporting a CLASS Statement

Procedures are classified as the "workhorses" in the SAS System. The CLASS statement specifies one, or more, character or numeric variables used to group data into classification levels. A virtue of using a CLASS statement is that a SORT procedure is not required to arrange and group the data, because the stats and other information is collected in memory and reported at the end of the procedure. A partial list of SAS procedures, below, supports the use of a CLASS statement.

SAS Procedures Supporting a CLASS Statement			
PROC ANOVA	PROC MEANS	PROC REPORT	PROC TTEST
PROC DISCRIM	PROC MIXED	PROC SUMMARY	PROC UNIVARIATE
PROC GENMOD	PROC NESTED	PROC SURVEYMEANS	
PROC GLM	PROC PHREG	PROC TABULATE	
PROC LOGISTIC	PROC REG	PROC TIMEPLOT	

Producing Page Numbers with ODS RTF Pagination Functions

Page numbering is the process of applying a sequence of numbers, Roman numerals, or letters on reports, spreadsheets, documents, books or other multi-page files. Legacy applications and program code frequently use counters or code routines to generate and display page numbers. Simple page numbering routines may resemble something similar to the following code.

DATA Step Code:

```
FILENAME REPORT DISK 'c:\DATA_NULL_Report.LST' ;
DATA _NULL_ ;
  SET SASHELP.CARS END=EOF ;
  FILE REPORT HEADER=H1 ; /* Execute Page_Header Routine */
  PUT @1 Origin  $6.
     @10 Make   $13.
     @25 MSRP   DOLLAR12. ;
RETURN ;

H1: ; /* Page Header */
  Page_CTR + 1 ;
  PUT @15 DATA _NULL_ Detail Report
   // @22 'Page Number ' Page_CTR  ;
RETURN ;
RUN ;
```

Page numbers can be produced and displayed in RTF output by specifying an escape character with an ODS RTF statement, any of the following functions, and an ODS RTF CLOSE ; statement:

- ✓ **{thispage}**
- ✓ **{lastpage}**
- ✓ **{pageof}**

Page Counters with ODS RTF Functions

Output Delivery System (ODS) provides powerful features that users can use when producing output. In the next example, an escape character is specified with the ODS RTF destination, where the functions: {thispage}, {lastpage}, and {pageof} are specified in the title and footnote statements to produce the page numbers and the total number of pages in the report.

ODS RTF Code:

```
ods escapechar='^' ;
ods RTF file='c:\Print-Report.rtf' ;
 proc print data=sashelp.cars noobs ;
   title 'Page ^{thispage} of ^{lastpage}' ;
   footnote '^{pageof}' ;
 run ;
ods RTF close ;
```

ODS RTF Results:

Page 1 of 24

'Page ^{thispage} of ^{lastpage}'

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2.0
Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4
Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2
Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5
Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5
Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2
Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8
Audi	A41.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8
Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3.0
Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	All	$33,430	$30,366	3.0
Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	All	$34,480	$31,388	3.0
Audi	A6 3.0 4dr	Sedan	Europe	Front	$36,640	$33,129	3.0
Audi	A6 3.0 Quattro 4dr	Sedan	Europe	All	$39,640	$35,992	3.0
Audi	A4 3.0 convertible 2dr	Sedan	Europe	Front	$42,490	$38,325	3.0
Audi	A4 3.0 Quattro convertible 2dr	Sedan	Europe	All	$44,240	$40,075	3.0
Audi	A6 2.7 Turbo Quattro 4dr	Sedan	Europe	All	$42,840	$38,840	2.7
Audi	A6 4.2 Quattro 4dr	Sedan	Europe	All	$49,690	$44,936	4.2
Audi	A8 L Quattro 4dr	Sedan	Europe	All	$69,190	$64,740	4.2
Audi	S4 Quattro 4dr	Sedan	Europe	All	$48,040	$43,556	4.2
Audi	RS 6 4dr	Sports	Europe	Front	$84,600	$76,417	4.2
Audi	TT 1.8 convertible 2dr (coupe)	Sports	Europe	Front	$35,940	$32,512	1.8
Audi	TT 1.8 Quattro 2dr (convertible)	Sports	Europe	All	$37,390	$33,891	1.8
Audi	TT 3.2 coupe 2dr (convertible)	Sports	Europe	All	$40,590	$36,739	3.2
Audi	A6 3.0 Avant Quattro	Wagon	Europe	All	$40,840	$37,060	3.0
Audi	S4 Avant Quattro	Wagon	Europe	All	$49,090	$44,446	4.2
BMW	X5 3.0i	SUV	Europe	All	$37,000	$33,873	3.0
BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4
BMW	325i 4dr	Sedan	Europe	Rear	$28,495	$26,155	2.5
BMW	325Ci 2dr	Sedan	Europe	Rear	$30,795	$28,245	2.5
BMW	325Ci convertible 2dr	Sedan	Europe	Rear	$37,995	$34,800	2.5
BMW	325xi 4dr	Sedan	Europe	All	$30,245	$27,745	2.5
BMW	330i 4dr	Sedan	Europe	Rear	$35,495	$32,525	3.0
BMW	330Ci 2dr	Sedan	Europe	Rear	$36,995	$33,890	3.0
BMW	330xi 4dr	Sedan	Europe	All	$37,245	$34,115	3.0
BMW	525i 4dr	Sedan	Europe	Rear	$39,995	$36,620	2.5
BMW	330Ci convertible 2dr	Sedan	Europe	Rear	$44,295	$40,530	3.0

1 of 24

'^{pageof}'

Automating the Process of Creating Multiple HTML Files

The Web offers incredible potential that impacts all corners of society. With its increasing popularity as a communications medium, Web publishers have arguably established the Web as the greatest medium ever created. Businesses, government agencies, professional associations, schools, libraries, research agencies, and a potpourri of society's true believers have endorsed the Web as an efficient means of conveying their messages to the world.

The SAS software provides users with the capability to create results and deploy selected pieces of output as HTML output files. Using the Output Delivery System (ODS) HTML destination, output can be created that anyone can view using a web browser. Syntactically correct HTML code is automatically produced and made ready for deployment using one of the Internet browser software products (e.g., Internet Explorer, Google Chrome, Mozilla FireFox, Safari, etc.). As a result, the SAS System and the HTML destination create a type of "streaming" or continuous output by adding elevator bars (horizontal and/or vertical) for easy navigation.

In the following example, redundant code and hardcoding issues are avoided by using PROC SQL to determine the number of unique (or distinct) values of the Origin column exist and once known are assigned to single-value and value-list macro variables. With the unique values assigned to two macro variables, an iterative %DO statement is specified to control the propagation of one, or more, HTML files containing one-way frequency results. The results of the three distinct HTML files that were created are also displayed, below.

ODS HTML Code:

```
/* Output HTML Files Location */
filename odsout "E:\" ;

options symbolgen ;
%macro multfiles ;
  proc sql noprint ;
    select count(distinct origin)
      into :morigin_cnt  /* derive number of origins    */
        from sashelp.cars
          order by origin ;
    select distinct origin
      into :morigin_list separated by "~" /* derive unique origin values */
        from sashelp.cars
          order by origin ;
  quit ;

  %do i=1 %to &morigin_cnt ;
    ods html path=odsout (URL=NONE)
             file="%SCAN(&morigin_list,&i,~)_FrequencyReport (MultiHTMLFiles).html"
            style=styles.barrettsblue ;
    title "Cars with Origin in %SCAN(&morigin_list,&i,~)" ;
    proc freq data=sashelp.cars(where=(origin = "%SCAN(&morigin_list,&i,~)")) ;
      tables type ;
      format msrp dollar12.0 ;
    run ;
    quit ;
    title ;
    ods html close ;
  %end ;
  %put &morigin_list ;
%mend multfiles ;

%multfiles ;
```

ODS HTML Results:

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Hybrid	3	1.90	3	1.90
SUV	25	15.82	28	17.72
Sedan	94	59.49	122	77.22
Sports	17	10.76	139	87.97
Truck	8	5.06	147	93.04
Wagon	11	6.96	158	100.00

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
SUV	10	8.13	10	8.13
Sedan	78	63.41	88	71.54
Sports	23	18.70	111	90.24
Wagon	12	9.76	123	100.00

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
SUV	25	17.01	25	17.01
Sedan	90	61.22	115	78.23
Sports	9	6.12	124	84.35
Truck	16	10.88	140	95.24
Wagon	7	4.76	147	100.00

Automating the Process of Creating Multiple Excel Files

Statistics show that the world's most used software application is Microsoft Excel®. Due to this dominance, SAS provides users with several ways to send results, tables, statistics, images and other output directly to an Excel spreadsheet. In the next example, redundant code and hardcoding issues are avoided by using PROC SQL to determine the number of unique (or distinct) values of the Origin column and, once known, are assigned to single-value and value-list macro variables. With the values assigned to the two macro variables, an iterative %DO statement is specified to control the propagation of Excel files containing one-way frequency results. The results of the three distinct Excel files that were created are also displayed, below.

ODS Excel Code:

```
%macro multExcelfiles ;
  proc sql noprint ;
    select count(distinct origin)
      into :morigin_cnt  /* derive number of origins    */
        from sashelp.cars
          order by origin ;
    select distinct origin
      into :morigin_list separated by "~" /* derive unique origin values */
        from sashelp.cars
          order by origin ;
  quit ;

  %do i=1 %to &morigin_cnt ;
    ods Excel file="e:/%SCAN(&morigin_list,&i,~)_FreqReport (MultiExcelFiles).xlsx"
            style=styles.barrettsblue ;
    title "Cars with Origin in %SCAN(&morigin_list,&i,~)" ;
    proc freq data=sashelp.cars(where=(origin = "%SCAN(&morigin_list,&i,~)")) ;
      tables type ;
      format msrp dollar12.0 ;
    run ;
    quit ;
    title ;
    ods Excel close ;
  %end ;
  %put &morigin_list ;
%mend multExcelfiles ;

%multExcelfiles ;
```

ODS Excel Results:

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Hybrid	3	1.90	3	1.90
SUV	25	15.82	28	17.72
Sedan	94	59.49	122	77.22
Sports	17	10.76	139	87.97
Truck	8	5.06	147	93.04
Wagon	11	6.96	158	100.00

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
SUV	10	8.13	10	8.13
Sedan	78	63.41	88	71.54
Sports	23	18.70	111	90.24
Wagon	12	9.76	123	100.00

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
SUV	25	17.01	25	17.01
Sedan	90	61.22	115	78.23
Sports	9	6.12	124	84.35
Truck	16	10.88	140	95.24
Wagon	7	4.76	147	100.00

Discovering the Number of Occurrences of a Value in a Data Set

Discovering the number of occurrences of individual values in a data set is useful information, particularly when constructing data-driven approaches. SAS provides several ways to count and determine the number of occurrences of a value in a data set.

Discovering the Number of Occurrences of a Value in a DATA Step

One approach for discovering the number of occurrences of a variable's value(s) is to construct a DATA step counting routine. In the next example, individual counters for the number of females and males are created, and after the last observation is read and processed, the results for each counter is output to the Counts data set, and the results displayed with PROC PRINT.

DATA Step Code:

```
data Counts(drop=Sex) ;
  set sashelp.Heart(keep=Sex) end=EOF ;
  if Sex = "Female" then Number_Females + 1 ;
  else if Sex = "Male" then Number_Males + 1 ;
  if EOF then do ;
     Total = Number_Females + Number_Males ;
     format Number_Females Number_Males Total comma7. ;
     output ;
  end ;
run ;
proc print data=Counts noobs ;
run ;
```

DATA Step Results:

Number_Females	Number_Males	Total
2,873	2,336	5,209

Discovering the Number of Occurrences of a Value with the PROC FREQ NLEVELS Option

Another approach for counting the number of occurrences of a variable's value(s) is to specify the NLEVELS option in PROC FREQ. In this example, the variable SEX is kept and the NLEVELS option is specified for the SASHELP.Heart data set. The results show there are two levels for the variable, SEX, with 2,873 females and 2,336 males.

PROC FREQ Code:

```
proc freq data=sashelp.Heart(keep=sex) NLEVELS ;
run ;
```

PROC FREQ Results:

The FREQ Procedure

Number of Variable Levels	
Variable	Levels
Sex	2

Sex	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Female	2873	55.15	2873	55.15
Male	2336	44.85	5209	100.00

Discovering the Number of Occurrences of a Value with PROC SQL

Another approach for counting the number of occurrences of a variable's value is to use the SUM function with an equality expression in PROC SQL. PROC SQL's data access and query capabilities offer SAS users with a powerful approach to summing down rows and across columns. In this example, a SELECT query is specified with a SUM function for counting the number of "Females", "Males" and their combined totals that are found in the SASHELP.HEART data set. An optional FORMAT=COMMA7. parameter is also specified to make the results easier to read.

PROC SQL Code:

```
proc sql ;
  select SUM(sex=”Female”) AS Number_Females format=comma7.,
         SUM(sex=“Male”)   AS Number_Males   format=comma7.,
         SUM(sex IN (“Female”,“Male”)) AS Total format=comma7.
    from sashelp.Heart ;
quit ;
```

PROC SQL Results:

Number_Females	Number_Males	Total
2,873	2,336	5,209

Using Metadata to Determine the Number of Observations in a Data Set

Metadata is everywhere and is defined as information that describes data. Other definitions include information about data, or information about the design and specification of objects and data structures. In its most basic form, metadata is found in the cataloging systems of every academic library, public library, school library, and special library in the world. The typical book, magazine, microfiche, digital file, image, or object’s metadata is stored in cataloging systems. These cataloging systems are not composed of words, sentences, paragraphs, or chapters, but contain information about its author(s), title, subject, keyword(s), description, publisher, publication date, ISBN, format, resource identifier, copyright, and other information.

Older Methods of Determining the Number of Observations in a Data Set

Before the availability of metadata in the SAS System, users developed and included code routines that determined the number of observations in a data set. An often used DATA step approach, since the beginning of SAS-time, constructs a variable that counts the number of observations. Although syntactically correct, this approach does not “scale” well – due to the amount of I/O incurred and the sizes of data sets – when computing the counter. The the next example, a DATA step approach computes the total number of “Sedans” found in the SASHELP.CARS data set, and displays the results using PROC PRINT.

DATA Step Code:

```
data sedans_counter(keep=type obs_ctr)
     cars_sedans(drop=obs_ctr) ;
  set sashelp.cars(keep=origin type make MSRP) end=eof ;
  where upcase(type) = “SEDAN” ;
  obs_ctr + 1 ;
  output cars_sedans ;
  if eof then output sedans_counter ;
run ;

proc print data=sedans_counter noobs ;
run ;
```

DATA Step Results:

Type	obs_ctr
Sedan	262

Using DICTIONARY.TABLES Metadata to Determine the Number of Observations in a Data Set

The SAS System collects and populates valuable metadata about SAS libraries, data sets (tables), catalogs, indexes, macros, system options, titles, views and other useful information in a collection of read-only tables called Dictionary tables. Dictionary tables serve a special purpose for SAS users by providing system-related information about the current SAS session's SAS databases and applications. When a query processes a Dictionary table, SAS automatically launches a discovery process at runtime to collect information pertinent to that table. This information is made available any time after a SAS session is started.

When users need more information about SAS data sets the TABLES Dictionary table can be very helpful. The TABLES Dictionary table provides detailed information about the library names, the member (or data set) names, the date a data set was created and last modified, the number of observations in a data set, and much more. The next example illustrates a popular approach that accesses the metadata content from the DICTIONARY.TABLES table to determine the number of observations in any SAS data set.

PROC SQL Code:

```
title "Number of Rows in a Table" ;

proc sql ;
  select libname, memname, nobs format=comma10.
    from Dictionary.Tables
      where nobs NE . ;
quit ;

title ;
```

PROC SQL Results:

Number of Rows in a Table

Library Name	Member Name	Number of Physical Observations
WORK	COUNTS	1
MYDATA	ACTORS	13
MYDATA	ACTORS_WITH_MESSY_DATA	15
MYDATA	AE	13
MYDATA	DM	24
MYDATA	EX	146
MYDATA	MOVIES	22
MYDATA	MOVIES_WITH_MESSY_DATA	31
SASHELP	AACOMP	2,020
SASHELP	AARFM	195
SASHELP	ADSMSG	426
SASHELP	AFMSG	1,090
SASHELP	AIR	144
SASHELP	APPLIANC	156
SASHELP	ASSCMGR	402
SASHELP	BASEBALL	322
SASHELP	BEI	24,205
SASHELP	BMIMEN	3,264
SASHELP	BMT	137
SASHELP	BURROWS	24,591
SASHELP	BUY	11
SASHELP	BWEIGHT	50,000
SASHELP	CARS	428
SASHELP	CITIDAY	1,069
SASHELP	CITIMON	145
SASHELP	CITIQTR	48
SASHELP	CITIWK	319
SASHELP	CITIYR	10
SASHELP	CLASS	19
SASHELP	CLASSFIT	19
SASHELP	QUAKES	15,578
SASHELP	RENT	10
SASHELP	RETAIL	58
SASHELP	REVHUB2	72
SASHELP	ROCKPIT	6
SASHELP	SASMBC	0
SASHELP	SASMSG	794
SASHELP	SHOES	395
SASHELP	SLKWXL	1,703
SASHELP	SMEMSG	34
SASHELP	SPRINGS	1,587
SASHELP	STEEL	44
SASHELP	STOCKS	699
SASHELP	STTMSG	336
SASHELP	SVRTDIST	2,373
SASHELP	SYR1001	105
SASHELP	TABLE	8
SASHELP	THICK	75
SASHELP	TIMEDATA	40,330
SASHELP	TOURISM	29
SASHELP	USECON	252
SASHELP	VBPLAYRS	11
SASHELP	VERBMGR	19
SASHELP	VIDMSG	7
SASHELP	VOTE1980	3,107
SASHELP	WEBMSG	349
SASHELP	WORKERS	67
SASHELP	YR1001	126
SASHELP	YR111	126
SASHELP	ZHC	7,445
SASHELP	ZIPCODE	41,232
SASHELP	ZTC	18,161
SASHELP	_CMPIDX_	44

Using SASHELP.VTABLE Metadata to Determine the Number of Observations in a Data Set

SAS also provides users with metadata content in a number of SASHELP views. In this next example the number of observations in any SAS data set can be determined by accessing the NOBS metadata content in the SASHELP.VTABLE view. This metadata content can be displayed using any output-producing SAS procedure, as shown below.

PROC PRINT Code:

```
title "Number of Rows in a Table" ;

proc print data=sashelp.vtable noobs ;
  var libname memname nobs ;
  format nobs comma10. ;
  where nobs NE . ;
run ;

title ;
```

PROC PRINT Results:

Number of Rows in a Table

libname	memname	nobs
WORK	COUNTS	1
MYDATA	ACTORS	13
MYDATA	ACTORS_WITH_MESSY_DATA	15
MYDATA	AE	13
MYDATA	DM	24
MYDATA	EX	146
MYDATA	MOVIES	22
MYDATA	MOVIES_WITH_MESSY_DATA	31
SASHELP	AACOMP	2,020
SASHELP	AARFM	195
SASHELP	ADSMSG	426
SASHELP	AFMSG	1,090
SASHELP	AIR	144
SASHELP	APPLIANC	156
SASHELP	ASSCMGR	402
SASHELP	BASEBALL	322
SASHELP	BEI	24,205
SASHELP	BMIMEN	3,264
SASHELP	BMT	137
SASHELP	BURROWS	24,591
SASHELP	BUY	11
SASHELP	BWEIGHT	50,000
SASHELP	CARS	428
SASHELP	CITIDAY	1,069
SASHELP	CITIMON	145
SASHELP	CITIQTR	48
SASHELP	CITIWK	319
SASHELP	CITIYR	10
SASHELP	CLASS	19
SASHELP	CLASSFIT	19
SASHELP	QUAKES	15,578
SASHELP	RENT	10
SASHELP	RETAIL	58
SASHELP	REVHUB2	72
SASHELP	ROCKPIT	6
SASHELP	SASMBC	0
SASHELP	SASMSG	794
SASHELP	SHOES	395
SASHELP	SLKWXL	1,703
SASHELP	SMEMSG	34
SASHELP	SPRINGS	1,587
SASHELP	STEEL	44
SASHELP	STOCKS	699
SASHELP	STTMSG	336
SASHELP	SVRTDIST	2,373
SASHELP	SYR1001	105
SASHELP	TABLE	8
SASHELP	THICK	75
SASHELP	TIMEDATA	40,330
SASHELP	TOURISM	29
SASHELP	USECON	252
SASHELP	VBPLAYRS	11
SASHELP	VERBMGR	19
SASHELP	VIDMSG	7
SASHELP	VOTE1980	3,107
SASHELP	WEBMSG	349
SASHELP	WORKERS	67
SASHELP	YR1001	126
SASHELP	YR111	126
SASHELP	ZHC	7,445
SASHELP	ZIPCODE	41,232
SASHELP	ZTC	18,161
SASHELP	_CMPIDX_	44

Using PROC PRINT with Style

Hecht (2011) describes the appearance of PROC PRINT output can be customized with colors, backgrounds, fonts, justifications, and other report components using styles. Styles can be specified for all destinations (e.g., RTF, PDF, HTML, Excel, etc.) except the Listing destination. In the next example, the SASHELP.CARS data set is sorted in ascending order by the variables Origin, Type, Make and MSRP; the HTML destination is opened with the HTMLBlue style selected for output; and background and foreground styles selected for the data, obs and total parts of the PROC PRINT report output.

PROC PRINT Code:

```
proc sort data=sashelp.Cars(keep=Origin Type Make MSRP)
          out=work.Cars_Sorted ;
  where MSRP < 20000 ;
  by Origin Type Make MSRP ;
run ;

ods HTML path="/folders/myfolders"  (url=none)
         file="PROC-PRINT-with-Style.html"
         style=HTMLBlue ;
proc print data=work.Cars_Sorted
  style (data)  = [background=Blue   foreground=white]
  style (obs)   = [background=red    foreground=white]
  style (total) = [background=yellow foreground=black] ;
  by Origin Type ;
  id Origin Type Make ;
  format msrp dollar12.0 ;
  sum MSRP ;
run ;
ods HTML close ;
```

PROC PRINT Results:

Origin	Type	Make	MSRP
Asia	Hybrid	Honda	$19,110

Origin	Type	Make	MSRP
Asia	SUV	Honda	$18,690
		Honda	$19,860
		Kia	$19,635
		Mitsubishi	$18,892
		Suzuki	$17,163
Asia	SUV		$94,240

Origin	Type	Make	MSRP
Asia	Sports	Hyundai	$18,739

Origin	Type	Make	MSRP
Asia	Truck	Mazda	$14,840
		Nissan	$19,479
		Toyota	$12,800
		Toyota	$16,495
Asia	Truck		$63,614

Origin	Type	Make	MSRP
Asia	Wagon	Kia	$11,905
		Mitsubishi	$17,495
		Scion	$14,165
		Suzuki	$16,497
		Toyota	$16,695
Asia	Wagon		$76,757
Asia			$974,429

Origin	Type	Make	MSRP
Europe	Sedan	MINI	$16,999
		MINI	$19,999
		Volkswagen	$18,715
		Volkswagen	$19,825
Europe	Sedan		$75,538

Origin	Type	Make	MSRP
Europe	Wagon	Volkswagen	$19,005
Europe			$94,543

Origin	Type	Make	MSRP
USA	Sports	Ford	$18,345

Origin	Type	Make	MSRP
USA	Truck	Chevrolet	$18,760
		Dodge	$17,630
		Ford	$14,385
		GMC	$16,530
USA	Truck		$67,305

Origin	Type	Make	MSRP
USA	Wagon	Ford	$17,475
		Pontiac	$17,045
USA	Wagon		$34,520
USA			$495,400
			$1,564,372

Using Available Memory with Hash Object Programming

Dorfman (2009) describes a SAS hash object as, "a high-performance look-up table residing completely in the DATA step memory." Due to the costs and availability of memory resources in today's computing environments, software vendors are doing everything they can to develop language constructs that capitalize on memory-resident operations. Dorfman further describes that, "The hash object is implemented via a Data Step Component Interface (DSCI), meaning that it is not a part of the DATA step proper. Rather, picture it as a black-box device you can manipulate from inside the DATA step to ask it for lightning-quick data storage and retrieval services."

Lafler (2016) describes a SAS hash object as, "a data structure that contains an array of items that are used to map identifying values, known as keys (e.g., employee IDs), to their associated values (e.g., employee names or employee addresses). As implemented, a hash object in the SAS System is used as a DATA step construct and is not available to any SAS Procedures." A hash object reads the contents of a data set into memory once allowing the SAS system to repeatedly access the data, as necessary. The contents of a hash object can be saved to a SAS data set (or table), but at the end of the DATA step the hash object and all its contents disappear. Since memory-based operations are typically faster than their disk-based counterparts, users often experience faster and more efficient table lookup, merge, sort and transpose operations.

Users with DATA step programming experience will find the hash object syntax relatively straight forward to learn and use. Available in all operating systems running SAS 9 or greater, the hash object is called using methods. The syntax for calling a method involves specifying the name of the user-assigned hash table, a dot (.), the desired method (e.g., operation) by name, and finally the specification for the method enclosed in parentheses. The following example illustrates the basic syntax for calling a method to define a key.

```
MatchTitles.DefineKey ('Title');
```

where MatchTitles is the name of the hash table, DefineKey is the name of the called method, and 'Title' is the specification being passed to the method.

In the next example, an essential operation frequently performed by users is the process of table lookup or search. The hash object as implemented in the DATA step provides users with the necessary tools to conduct match-merges (or joins) of two or more data sets. Data does not have to be sorted or be in a designated sort order before use as it does with the DATA step merge process. The following code illustrates a hash object with a simple key (TITLE) to merge (or join) the MOVIES and ACTORS data sets to create a new data set (MATCH_ON_MOVIE_TITLES) with matched observations. *The same operation is handled in a DS2 section example here*.

Hash Object Code:

```
data match_on_movie_titles(drop=rc) ;

❶ if 0 then set mydata.movies
                mydata.actors ;  /* load variable properties into hash tables */

  if _n_ = 1 then do ;
❷    declare Hash MatchTitles (dataset:'mydata.actors') ; /* declare the name
                                                          MatchTitles for hash */

❸    MatchTitles.DefineKey ('Title') ; /* identify variable to use as key */
     MatchTitles.DefineData ('Actor_Leading',
                             'Actor_Supporting') ; /* identify columns of data */
     MatchTitles.DefineDone () ; /* complete hash table definition */
  end ;

  set mydata.movies ;

❹ if MatchTitles.find(key:title) = 0 then output ; /* lookup TITLE in MOVIES
                                                    table using MatchTitles hash */
run ;
```

Hash Object Results:

	Title	Length	Category	Year	Studio	Rating	Actor_Leading	Actor_Supporting
1	Brave Heart	177	Action Adventure	1995	Paramount Pictures	R	Mel Gibson	Sophie Marceau
2	Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13	Chevy Chase	Beverly D'Angelo
3	Coming to America	116	Comedy	1988	Paramount Pictures	R	Eddie Murphy	Arsenio Hall
4	Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13	Tom Hanks	Sally Field
5	Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13	Patrick Swayze	Demi Moore
6	Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R	Mel Gibson	Danny Glover
7	Michael	106	Drama	1997	Warner Brothers	PG-13	John Travolta	Andie MacDowell
8	National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13	Chevy Chase	Beverly D'Angelo
9	Rocky	120	Action Adventure	1976	MGM / UA	PG	Sylvester Stallone	Talia Shire
10	Silence of the Lambs	118	Drama Suspense	1991	Orion	R	Anthony Hopkins	Jodie Foster
11	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG	Sean Connery	Alec Baldwin
12	The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R	Arnold Schwarzenegge	Michael Biehn
13	Titanic	194	Drama Romance	1997	Paramount Pictures	PG-13	Leonardo DiCaprio	Kate Winslet

User-developed Macros

There are 2 kinds of user developed macros, single use "through away" code and special purpose, multi-use macros. The latter should be stored in a central repository (or directory) for a particular project, department or an entire organization. These macros should be completely documented from top to bottom. Including an explanation of the macros purpose, its input parameters and expected output along with a sample test run, as shown in the example below. (*A FCMP Subroutine that accomplishes the same character stripping but for a datastep variable is found here*) A contents readme file documenting the macros in the collection should be kept in the directory with the macros so users can see what's available. For more information about the SAS macro language, see Carpenter's Complete Guide to the SAS® Macro Language, Third Edition.

MSTRIPCHR Macro:

```
/* ************************************************************************************
 * MACRO: MSTRIPCHR removes a specified character recursively from the beginning or
 *        end of a given macro character string. %, &, ' and " are not allowed for
 *        any resolved parameter value and will cause an ERROR!!!
 *
 * By:       Clark Roberts, Decision Analytics
 *
 * Updates:  2003    Program written.
 *           2017    Added _LOC parameter and logic to
 *                   strip _CHR from the end of _STR
 *
 * Copyright Decision Analytics 2003 - 2017
 * All rights reserved
 *
 ************************************************************************************ */

 %macro mstripchr(_str ,    /* <REQUIRED> */
                  _chr ,    /* <REQUIRED> */
                  _n   ,    /* <REQUIRED> */
                  _loc = S
                 )
                 ;

/* ************************************************************************************
 *     PARAMETERS: _STR: Name of a macro variable within the scope of the calling
 *                       program that contains the string of characters to be
 *                       processed. After the macro executes any leading characters
 *                       specified in the _CHR parameter.
 *                 _CHR: Name of a macro variable within the scope of the calling
 *                       program that contains the character to be stripped. It's
 *                       value does not change as a result of the macro execution.
```

```
*                 _N:   Name of a macro variable within the scope of the calling
*                       program that will contain the number of _CHR values that
*                       were removed.  It must be initialized to 0 (zero) before
*                       calling STRIPCHR.
*                 _LOC  Location of the character (_CHR) to be stripped from
*                       (_STR). S -> beginning (default), E -> end.
********************************************************************************* */

   %let _loc = %upcase(%substr(&_loc,1,1));
   %if &_loc ^= E and &_loc ^= S %then %let _loc = S;
   %if &_loc = E %then %let &_str = %qsysfunc(reverse(%nrbquote(&&&_str)));
   %let &_chr = %qsubstr(%nrbquote(&&&_chr),1,1);
   %if %length(%nrbquote(&&&_str)) > 1 %then %do;
       %if %qsubstr(%qtrim(%qleft(%nrbquote(&&&_str))),1,1) = %nrbquote(&&&_chr)
        %then %do;
          %let &_n = %eval(&&&_n + 1);
          %let &_str = %qsubstr(%nrbquote(&&&_str),2);
          %mstripchr(&_str, &_chr, &_n, _loc=S);
       %end;
   %end;
   %if &_loc = E %then %let &_str = %qsysfunc(reverse(&&&_str));

%mend mstripchr;
```

MSTRIPCHR Macro Log Results:

```
66         * options symbolgen macrogen mlogicnest source source2;
67          %macro testmstripchr();
68          %local var chr loc n ;
69          %let n = 0;
70          %let var = ----------------------------------------testval2;
71          %let chr = -;
72          %let loc = s;
73          %put before mstripchr called &n &var &chr &loc;
74          %mstripchr(var, chr, n, _loc=&loc);
75          %put after mstripchr called &n &var &chr &loc;
76          %let var = testval3::::::::::::::::::::::::::::;
77          %let chr = :;
78          %let loc = e;
79             %let n = 0;
80             %put before mstripchr called &n &var &chr &loc;
81          %mstripchr(var, chr, n, _loc=&loc);
82          %put after mstripchr called &n &var &chr &loc;
83          %mend testmstripchr;
84          %testmstripchr();

before mstripchr called 0 ----------------------------------------testval2 - s
after mstripchr called 38 testval2 - s
before mstripchr called 0 testval3:::::::::::::::::::::::::::: : e
after mstripchr called 27 testval3 : e
```

PC Windows Environment

SAS provides the MODULExy family of call routines and functions (including MODULE, MODULEN, MODULEC, MODULEI, MODULEIN, and MODULEIC) to access system related API functions and other DLL function calls. It's a very powerful tool, but somewhat obscure so it's not widely used currently.

The use of these SAS MODULExy functions requires creation of a text file (SASCBTBL) that describes the API/DLL routine to be invoked. This file can be permanently created or created in-line with FILE and PUT statements, as the following example demonstrates. Download the Win32 API Declarations for Visual Basic reference or for a version updated for 64 bit office go to this Microsoft Office site. Scroll down and, under the "**Links**" section, click the link associated with "Office 2010 Help Files:

Win32API_PtrSafe with 64-bit Support" to automatically download the Win32API_PtrSafe with 64-bit Support. Then, run the downloaded EXE file to install the TXT and related files. These references will give you the information you need to help build the SAS SASCBTBL file. Many of the 32 bit Win32API functions will work in a 64 bit environment without modification (refer to Langston 2015 for information on MODULExy 64 bit usage). It seems that pointers are the main issue and the above reference to the Win32API_PtrSafe functions should hopefully resolve any of those issues.

In the following example, the OPENWORD macro opens Microsoft Word for a DDE session. It uses the Win32API calls "FindExecutableA" to find the location of Word for Windows (WinWord) and "WinExec" to execute WinWord. The authors are using the SAS University Edition which runs in a Linux virtual machine, but this macro needs direct access to Windows so an example of its use was not possible. There are a number of other good examples in the articles in the WIN32API References and Suggested Reading section.

MODULExy functions are available in other operating systems (eg. UNIX, VMS, etc.). Refer to the SAS Companion for these operating systems for details.

MODULEN Function Example Code:

```
*******************************************************************************
*** Source:     OPENWORD.SAS (MS Word 95/97 under Win95/98 SAS 6.12 Product
Version)
***
*** Type:       Include Module
***
*** Function:   Opens MS Word for a SAS to Word DDE session
***
*** Note:       This version has been tested in MS Word 95 and 97 under Windows 95
***             and 98 in SAS 6.12 (TS025 and above)
***
*** Parameters: None
***
*** By:         Clark Roberts, Decision Analytics
***
*** Updates:    5-30-97 Program written.
***             10-7-99 Added logic to build the SAS modulen control table
***                     in the OPENWORD macro.
***
***             Copyright Decision Analytics 1997 - 2017.
***             All rights reserved
*******************************************************************************;

%macro openword();

   ****************************************************************************
   *** Define a global macro variable for the return code to be tested by the
   *** calling program.  If the operation is successful then the return code
   *** will be > 0, If it fails then the return code will be 0.
   ****************************************************************************;

   %global rcword;

   %let rcword = 1;

   ****************************************************************************
   ***
   *** Define the parameter settings for the Win API calls that will be used by
   *** the SAS modulen function for this program. These can be stored permanently
   *** in a text file for better performance, if required. Also create an empty
   *** .DOC file for the FindExecutableA API call to use to find the path to the
   *** MS Word executable WINWORD.EXE.
   ***
   *** Win32API Visual Basic Declarations
   *** --------------------------------
   ***
```

```
*** Declare Function WinExec Lib "kernel32" Alias "WinExec"
***    (ByVal lpCmdLine As String, ByVal nCmdShow As Long)
***    As Long
***
*** Declare Function FindExecutable Lib "shell32.dll" Alias
***     "FindExecutableA"(ByVal lpFile As String,
***     ByVal lpDirectory As String, ByVal lpResult As String)
***     As Long
***
*******************************************************************************;

filename modtabl 'c:\modtabl.txt';

data _null_;

     file modtabl;

     put @2 'routine WinExec';
     put @5 'maxarg=2';
     put @5 'minarg=2';
     put @5 'stackpop=called';
     put @5 'module=KERNEL32';
     put @5 'returns=ushort;';
     put @7 'arg 1 input char format=$cstr200.;';
     put @7 'arg 2 input num byvalue format=pib4.;' //;

     put @2 'routine FindExecutableA';
     put @5 'maxarg=3';
     put @5 'minarg=3';
     put @5 'stackpop=called';
     put @5 'module=shell32';
     put @5 'returns=ushort;';
     put @7 'arg 1 input char format=$cstr200.;';
     put @7 'arg 2 input char format=$cstr200.;';
     put @7 'arg 3 output char format=$cstr200.;';

run;

filename modtabl clear;

filename wordtemp 'c:\system.doc';

data _null_;
     file wordtemp;
     put 'this is a dummy .doc file for the FindExecutableA API';
     put 'to use to locate the MS Word .EXE file.  It is created';
     put 'by the SAS OPENWORD macro';
run;

filename wordtemp clear;

*******************************************************************************
*** If MS Word is already open, then close it and allow the user to save
*** their work.
*******************************************************************************;

filename wordchk dde 'winword|system!selected' command;

%let status = %sysfunc(fopen(wordchk,S));

%if &status %then %do;
   data _null_;
           file wordchk;
           put '[FileExit 1]';
   run;
```

```
      %let rc = %sysfunc(fclose(&status));
      filename wordchk clear;
   %end;

   ********************************************************************************
   ***   Start a MS Word session if requested.  SAS calls to the Windows APU are
   ***   used to find the location of MS Word and if found, starts a session.
   ***   The modtabl.txt file, which contains the parameters for each system call
   ***   and the system.doc file, which is used as a trigger for the
   ***   FindExecutableA API function to locate MS Word on the system. The SAS
   ***   MODULEN function is used for the API interface. If MS Word fails to open,
   ***   then set the RCWORD return code to 0,  The Word session is started in
   ***   minimized state.
   ********************************************************************************;

   filename sascbtbl "c:\modtabl.txt";

   data _null_;
        length sysdir wordpath $200;
        wordpath = '';
        sysdir   = 'c:\';
        rc1 = modulen('*ei','FindExecutableA','system.doc',sysdir,wordpath);
        if rc1 > 32 then do;
           rc2 = modulen('*ei','WinExec',wordpath,2);
           if rc2 <= 0 then call symput('rcword','0');
        end;
        else do;
           put 'OPENWORD ERROR: Could not find the MS Word executable';
           call symput('rcword','0');
        end;
        put rc1= rc2= wordpath=;
   run;

   filename sascbtbl clear;

   filename wordsys dde 'Winword|System';

%mend openword;

/*************************************************************
***                  E N D   O F   M O D U L E
***                        OPENWORD.SAS
*************************************************************/
```

User-developed Functions with the FCMP and PROTO Procedures

In version 9.2, SAS added the ability to create user-written functions and call routines, utilizing the SAS DATA step and DS2 programming languages, using the PROC FCMP and its cousin, PROC PROTO. PROC PROTO is used to incorporate C language functions into SAS, while PROC FCMP compiles SAS DATA Step and DS2 language elements for use in those environments. "FCMP routines are stored in a data set and can be called from several SAS/STAT®, SAS/ETS® , and SAS/OR® software procedures, like the NLIN, MODEL, and NLP procedures. In SAS 9.2, FCMP routines can be called from a DATA step." (Secowski 2007).

User-written routines are defined using PROC FCMP and can be either a function or a CALL routine. A function is a subroutine that returns a value and accepts either zero or more arguments passed by value and will not modify the value in the calling routine. On the other hand a CALL routine is a subroutine that does not return a value explicitly and normally receives one or more arguments passed by value, or also by reference. When passed by reference one copy of the data is shared between the calling routine and the subroutine. This is how information is returned to the calling program. The next example illustrates a user-defined function and its related subroutine that converts yards to meters. The function is contained with a larger collection of functions stored in the ***ExampleFunctions*** library. *To view the use of these functions in a DS2 package refer here.*

FCMP User-defined Function and Subroutine:

```
libname cr "/folders/myfolders/CR" ; /* permanently store functions in*/
                                     /* ExampleFunctions library for  */
                                     /* reuse                         */

/*********************************************************************
  Functions and Subroutines to convert various units of measurement
*********************************************************************/
options cmplib = cr.ExampleFunctions ;

proc fcmp outlib=cr.ExampleFunctions.Yards2Meters ;

function Yards2Meters(inUnits) ;
   return(inUnits / 0.9144) ;
endsub ;

proc fcmp outlib=cr.ExampleFunctions.Yards2Meters_sub ;

subroutine Yards2Meters_sub(inUnits, outUnits);
   outargs outUnits ;
   outUnits = Yards2Meters(inUnits) ;
   return ;
endsub ;

proc fcmp outlib=cr.ExampleFunctions.Inches2Centimeters ;

function Inches2Centimeters(inUnits) ;
   return(inUnits * 2.54) ;
endsub ;

proc fcmp outlib=cr.ExampleFunctions.Centimeters2Inches ;

function Centimeters2Inches(inUnits) ;
   return(inUnits / 2.54) ;
endsub ;

proc fcmp outlib=cr.ExampleFunctions.Kilograms2Pounds ;

function Kilograms2Pounds(inUnits) ;
   return(inUnits * 2.2046226218) ;
endsub ;

quit ;

/*********************************************************************
    Test program for yards to meters function and subroutine
*********************************************************************/
data _null_;
  length invar outvar 8;
  invar = 5;
  outvar = Yards2Meters(invar);
  put "function call " outvar=;
  invar = 5;
  call Yards2Meters_sub(invar, outvar);
  put "subroutine call " outvar= ;
run;
```

FCMP User-defined Function and Subroutine Log Results:

```
 84  /*********************************************************
 85    Test program for yards to meters function and subroutine
```

```
86  ************************************************************/
87  data _null_;
88    length invar outvar 8;
89    invar = 100;
90    outvar = Yards2Meters(invar);
91    put "function call " outvar=;
92    invar = 100;
93    call Yards2Meters_sub(invar, outvar);
94    put "subroutine call " outvar= ;
95  run;

function call outvar=109.36132983
subroutine call outvar=109.36132983

NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

PROC FCMP routines can be recursive. Sekowski (2007) describes "Recursion as a problem-solving technique that reduces a problem to a smaller one that is simpler to solve and then combines the results of the simpler solution(s) to form a complete solution. A recursive function is a function that calls itself, either directly or indirectly." The next example demonstrates the concept of recursion by using the character stripping paradigm, only this time a PROC FCMP subroutine is used to modify a DATA step variable instead of a macro variable *(refer to the earlier macro example in the User Developed Macros section*). The next example illustrates a user-defined FCMP routine and the result from the test program is illustrated, below.

Recursive User-defined FCMP Routine:

```
/*****************************************************************
   FCMP subroutine to recursively strip character _CHR from the
   Beginning or end of a character string _STR
*****************************************************************/

options cmplib = work.ExampleFunctions;

proc fcmp outlib=work.ExampleFunctions.stripchr;

subroutine stripchr(_str $, _chr $, _n , _loc $, origloc $);
    outargs _str, _chr, _n, _loc, origloc;
    _loc = upcase(substr(_loc,1,1));
    if _loc ^= 'E' and _loc ^= 'S' then do;
       _loc = 'S';
       origloc = _loc;
    end;
    if _loc = 'E' then do;
       _str = trim(left(reverse(_str)));
       _loc = "S";
    end;
    _chr = substr(_chr,1,1);
    if length(_str) > 0 then do;
        if substr(trim(left(_str)),1,1) = _chr then do;
           _n = _n + 1;
           _str = substr(_str,2);
           call stripchr(_str, _chr, _n, _loc, origloc);
           return;
        end;
    end;
    if upcase(origloc) ^= 'S' then do;
       _str = reverse(_str);
       _loc = "E";
```

```
    end;
   return;
endsub;
quit;
```

Test Program for the STRIPCHR FCMP Function:

```
/*************************************************
* Test program for stripchr FCMP subroutine
*************************************************/
data before after ;
   length _str $200 _chr $5 _loc $5 _n 4 ;

   _chr = '-'  ;
   _loc = 'x' ;
   _str = '----------------------------------------------testval1' ;
   link process ;

   _chr = ">" ;
   _loc = "e" ;
   _str = "testval2>>>>>>>>>>>>>>>>>>" ;
   link process ;

   _chr = "$" ;
   _loc = "s" ;
   _str = "$$$$$$$$$$$$$$$$$testval3" ;
   link process ;

   _chr = "?" ;
   _loc = "e" ;
   _str = "testval4??????????????????????????" ;
   link process ;

   _chr = "." ;
   _loc = "s" ;
   _str = ".....................................testval5" ;
   link process ;

   goto exit ;

   process:
     output before ;
     _n = 0 ;
     origloc = upcase(substr(_loc,1,1)) ;
     call stripchr(_str, _chr, _n, _loc, origloc) ;
     output after ;
     return ;

   exit :
   run ;

   proc print data=before ;
     var _str _chr _loc ;
     title "Values before function call" ;
   run ;

   proc print data=after ;
     var _str _n _loc ;
     title "Values after function call" ;
   run ;
```

Recursive User-defined FCMP Routine Results:

Values before function call

Obs	_str	_chr	_loc
1	----------testval1	-	x
2	testval2>>>>>>>>>>>>>>>>>>	>	e
3	$$$$$$$$$$$$$$$$$testval3	$	s
4	testval4????????????????????????????	?	e
5	.....testval5	.	s

Values after function call

Obs	_str	_n	_loc
1	testval1	47	S
2	testval2	19	E
3	testval3	17	S
4	testval4	27	E
5	testval5	39	S

PROC FCMP provides the special function named RUN_MACRO that enables a user-written routine to run a macro. There is no restriction as to what the macro can generate in terms of SAS code: it can include a number of DATA steps and PROC steps. Since a user-written routine is callable from within a DATA step then RUN_MACRO makes it's possible to have nested DATA steps. References by Carpenter (2016) and Deguire (2013) provide more detail on the use of RUN_MACRO.

Beginning with SAS® 9.3, hashing became available to user-defined subroutines through the FCMP Procedure. "Hashing is an important construct often used to improve performance of operations such as merging, filtering and searching. Hashing has been supported in the DATA step for more than a decade... Subroutines have always encapsulated and modularized code, making programs reusable, and the addition of hashing allows users to extend the scope of their programs, tackling larger problems without sacrificing simplicity." (Henrick 2017). For more information about the DATA step hash construct, see Lafler (2016) and Dorfman / Henderson (2017).

PROC DS2

Introduced in SAS Version 9.4 in 2013, DS2 is a data step like programming language redesigned from the ground up with the following goals in mind:

- Process ANSI SQL natively for integration with external data sources.
- Provide syntax for multi-threaded processing for accelerated execution of CPU intensive tasks.
- Provide just enough Object Oriented Programming (OOP) constructs, including method overloading, for reusability of DS2 code. Method overloading consists of having 2 or more methods with the same name, but with different signatures. Signatures consist of the methods name and ordered list of parameter data classes.

The differences between the data step and DS2 are:

- DS2 supports ANSI native numeric data types and multi-byte character types to directly access DBMS data.
- DS2 is multi-threaded whereas the data step is single threaded providing acceleration for CPU bound processing by DS2. If the process is I/O bound the multi-threading in DS2 will not help reduce run times.
- DS2 programs can take advantage of the SAS In-Database Code Accelerator by allowing DS2 programs to execute on the Massively Parallel Processing (MPP) capabilities on the DBMS hardware with the SAS Embedded Process.
- There are almost no reserved words in the data step language. In DS2 all keywords are reserved word
- Variables referenced in a data step are global in scope, Variables referenced in a DS2 program can be either global or local in scope.
- All variables referenced in the data step are in the Program Data Vector, while DS2 variables of local scope are not in the PDV.
- Creating reusable data step code with variable encapsulation requires PROC FCMP. Reusable code in DS2 with variable encapsulation is possible using PROC DS2 syntax in a 'package' program.

- The data step can consume a table produced by an SQL query as input to a set statement. While DS2 can directly accept the result set of an SQL query as input to the set statement.

Some of the prerequisites for using DS2 include:

- DATA step programming experience
- Accessing data with a LIBNAME statement
- The role of the program data vector (PDV) in DATA step processing
- Conditional processing with IF-THEN-ELSE, SELECT-WHEN, etc. statements
- DO loop processing
- Array processing
- SAS Macro language
- SQL

Within a DS2 block you can define and execute three types of program blocks:

Program Block	Brief Description
Data	The heart of the DS2 language, data programs manipulate input datasets to produce output result sets. They can receive input from tables, SQL result sets or thread program result sets.
Package	A package programming block or package refers to the stored library of variables and methods bounded by PACKAGE…ENDPACKAGE statements. The variables and methods of a package can be used by DS2 programs, threads, or other packages, enabling an object oriented approach to development.
Thread	Thread programs manipulate input datasets to produce output result sets that are returned to a data program. Used to simultaneously process several rows of data in parallel threads. A thread programming block, or thread program, refers to a stored program that is bounded by the THREAD…ENDTHREAD statements. The thread program can be called by the SET FROM statement in a DS2 program or package.

Methods

A method programming block or method block refers to a sub-block of programming statements that are bounded by the in methods. METHOD and END statements. User methods need to be defined before the 3 internal built-in methods INIT, RUN and TERM. The INIT method runs first where any initializations takes place. The RUN method executes next and finally the TERM method runs last. If the user doesn't define these then DS2 automatically creates them. User methods can be called from any of the internal built-packages.

Packaging

A package is a collection of methods and variables that can be used in DS2 programs. "… packages are analogous to classes in other object-oriented languages such as C# or Ruby and can be used to massively improve your programming effectiveness." (*Brooks 2016*) Packages should be stored in a data set within a SAS library for re-use. The first example below provides an example of defining and storing a package. To use a package; a DS2 program, another package, or a thread instantiates the package to access its methods.

"With the addition of user-defined packages to the SAS DS2 programmer's toolbox SAS have opened up a new way of designing and programming our SAS applications. If we, as SAS developers, use this facility in a true object-oriented way we can significantly improve code re-use, encourage standardization of techniques and remove much of the routine coding which is part and parcel of the traditional procedural programming paradigm." (*Brooks 2016*)

Built-in Packages (refer to *Jordan 2016* for detailed coverage of built-in packages):

SQLSTMT Package

Provides a way to pass FedSQL statements to a DBMS for execution and to access the result set returned by the DBMS. There are performance benefits of using DS2 over PROC SQL. "*The SQLSTMT package and the SQLEXEC function enable*

DS2 programs to dynamically generate, prepare, and execute FedSQL statements to update, insert, or delete rows from a table. With an instance of the SQLSTMT package or the SQLEXEXC function, the FedSQL statement allocate, prepare, execute, and free occurs at run time." (Kaufmann 2014)

MATRIX Package

The MATRIX package provides you with a powerful and flexible matrix programming capability.

HTTP and JSON Packages

The HTTP package provides the vehicle to construct an HTTP client to retrieve and post data on the internet. You can also retrieve status codes to log traffic between the client and server using the SAS Logging facility.

The JSON (JavaScript Object Notation) package gives you a means for creating and parsing JSON text.

HASH AND HASH ITERATOR Packages

These packages provide data hashing capability to DS2 routines like their data step equivalents.

FCMP Package

The Function Compiler procedure (FCMP) supports calls to FCMP functions and subroutines from within the DS2 language.

TIMEZONE (TZ) Package

The TZ package allows DS2 to process ANSI date, time and timestamp values from different time zones more easily.

LOGGER Package

Provides a basic interface (open, write, and level query) to the SAS logging facility.

The first example uses the SET statement of the SQLSTMT package. *This is the same example as the DATA step version demonstrated earlier*. In the case of the SET statement, no package instantiation is required. The SASHELP cars dataset is subset on the variable TYPE for records containing SUV, Truck or Wagon. For some reason SAS didn't like the SASHELP libname in the DS2 block, so we created a copy in WORK first.

DS2 SQLSTMT Package Example:

```
data cars;
  set sashelp.cars;
run;

proc ds2;
 Data work.vehicles / overwrite=yes;
    method run();
      set  {select origin, type, msrp from work.cars
            where upcase(type) in ('SUV', 'TRUCK', 'WAGON')
            order by msrp};
    end;
 enddata;
 run;
quit;

proc print data=vehicles(obs=17);
  format msrp dollar8.;
run;
```

Log Results from Package Example:

```
62         data cars;
63              set sashelp.cars;
64         run;

NOTE: There were 428 observations read from the data set SASHELP.CARS.
NOTE: The data set WORK.CARS has 428 observations and 15 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

65
66         proc ds2;
67          Data work.vehicles / overwrite=yes;
68             method run();
69               set  {select origin, type, msrp from work.cars
70                     where upcase(type) in ('SUV', 'TRUCK', 'WAGON')
71                     order by msrp};
72             end;
73          enddata;
74          run;
NOTE: Execution succeeded. 114 rows affected.
75         quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time           0.05 seconds
      cpu time            0.05 seconds

76
77         proc print data=vehicles(obs=17);
78          format msrp dollar8.;
79         run;

NOTE: There were 17 observations read from the data set WORK.VEHICLES.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.05 seconds
      cpu time            0.05 seconds
```

Partial PROC PRINT Output from Program:

Obs	ORIGIN	TYPE	MSRP
1	Asia	Wagon	$11,905
2	Asia	Truck	$12,800
3	Asia	Wagon	$14,165
4	USA	Truck	$14,385
5	Asia	Truck	$14,840
6	Asia	Truck	$16,495
7	Asia	Wagon	$16,497
8	USA	Truck	$16,530
9	Asia	Wagon	$16,695
10	USA	Wagon	$17,045
11	Asia	SUV	$17,163
12	USA	Wagon	$17,475
13	Asia	Wagon	$17,495
14	USA	Truck	$17,630
15	Asia	SUV	$18,690
16	USA	Truck	$18,760
17	Asia	SUV	$18,892

The second example is an example of the use of the HASH package that duplicates the datastep merge example in *DATA step hash example* merging leading actors and supporting actors with their corresponding movie titles to provide additional information. The key, data and dataset name, etc. is entered with the package instantiation statement instead of separate statements for each of these to reduce the lines of code. The additional fields in the instantiation statement are hashexp (1), ordered (a for ascending), action for duplicate key entries (error) and multidata (no}.

DS2 Hash Package Example:

```
libname mydata '/folders/myfolders/CR';
proc ds2;
  data match_on_movie_titles / overwrite=yes;
    dcl varchar(20) ActorLeading ActorSupporting;
    dcl varchar(30) Title;
         /* Define Hash Table MatchTitles */
    dcl package hash MatchTitles ([Title],[ActorLeading ActorSupporting],1,
              'mydata.actors', 'a', 'error', '', 'no');
   Method run();
        set mydata.movies ;
              /* lookup TITLE in MOVIES table using MatchTitles hash */
        if MatchTitles.find() = 0;
       End;
  enddata;
run;
quit;
proc print data=match_on_movie_titles;
 var Title Length Category Year Studio Rating ActorLeading ActorSupporting;
run;
```

Output from DS2 Hash Package Example:

Obs	Title	Length	Category	Year	Studio	Rating	ActorLeading	ActorSupporting
1	Brave Heart	177	Action Adventure	1995	Paramount Pictures	R	Mel Gibson	Sophie Marceau
2	Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13	Chevy Chase	Beverly D'Angelo
3	Coming to America	116	Comedy	1988	Paramount Pictures	R	Eddie Murphy	Arsenio Hall
4	Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13	Tom Hanks	Sally Field
5	Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13	Patrick Swayze	Demi Moore
6	Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R	Mel Gibson	Danny Glover
7	Michael	106	Drama	1997	Warner Brothers	PG-13	John Travolta	Andie MacDowell
8	National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13	Chevy Chase	Beverly D'Angelo
9	Rocky	120	Action Adventure	1976	MGM / UA	PG	Sylvester Stallone	Talia Shire
10	Silence of the Lambs	118	Drama Suspense	1991	Orion	R	Anthony Hopkins	Jodie Foster
11	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG	Sean Connery	Alec Baldwin
12	The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R	Arnold Schwarzenegge	Michael Biehn
13	Titanic	194	Drama Romance	1997	Paramount Pictures	PG-13	Leonardo DiCaprio	Kate Winslet

User Defined Packages

- These are packages that you can use to store methods for any purpose. A user can store methods that they create in user-defined packages.
- These packages can be thought of as libraries of your methods. Any type of method can be saved in a package. Once you have stored methods in a package (using the PACKAGE statement), you can access them by creating an instance of the package with only a DECLARE statement or with the _NEW_ operator.

The following example is taken from the *first FCMP example*, the stored functions in ExampleFunctions in the CR library. The wrapper routine creates a package containing the stored functions for use in DS2. The test program instantiates the package as ConvertIt and calls the Kilograms2Pounds function.

By storing the "function library" as a package the routines can then be used in both a DATA step program and data program in DS2. A little confusing at first, but eventually it'll become clearer.

DS2 FCMP Wrapper Program for Permanent Function Storage:

```
/* Wrap the FCMP functions in a DS2 package */
libname cr "/folders/myfolders/CR" ;
proc ds2;
    package cr.fcmp_converters / overwrite=yes language='fcmp'
                table='cr.ExampleFunctions' ;
run;
quit;
```

Test Program for DS2 FCMP Function Call:

```
/************************************************************
    Test program for unit conversion FCMP DS2 package
************************************************************/
proc ds2;
   data _null_;
      dcl package cr.fcmp_converters convertIt();
      dcl double invar having format 12.6;
      dcl double outvar having format 12.6;
      method run();
         invar=13.6;
         outvar = convertIt.Kilograms2Pounds(invar);
         put 'Kilograms2Pounds ' invar= outvar= ;
      end;
   enddata;
   run;
quit;
```

Log Output:

```
62          /************************************************************
63              Test program for unit conversion FCMP DS2 package
64
65          ************************************************************/
66          proc ds2;
67          data _null_;
68
69          dcl package cr.fcmp_converters convertIt();
70             dcl double invar having format 12.6;
71             dcl double outvar having format 12.6;
72
73             method run();
74             invar=13.6;
75             outvar = convertIt.Kilograms2Pounds(invar);
76             put 'Kilograms2Pounds ' invar= outvar= ;
77             end;
```

```
78
79            enddata;
80         run;

Kilograms2Pounds  invar=   13.600000 outvar=   29.982868

NOTE: Execution succeeded. No rows affected.
81         quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time           0.39 seconds
      cpu time            0.23 seconds
```

Introduced in SAS® 9.3 the High-performance version of the DS2 procedure, PROC HPDS2, executes DS2 language statements in a SAS High-Performance Analytics environment for parallel execution. It is an efficient method for moving big data to the servers or network workstations for distributed processing. PROC HPDS2 is most useful when significant amounts of computationally intensive, row-independent logic must be applied to the data. The HPDS2 procedure offers the following features:

- Enables DS2 code to be executed on a local client machine or on the SAS High-Performance Analytics grid
- Enables control of the level of parallelism per execution node and the number of nodes to engage
- Performs a syntax check of the DS2 code on the local client machine before sending it to the grid for execution
- Manages data migration to the location of execution and movement back to the client machine as needed
- Runs within the framework of SAS Embedded Process

Parallel Processing

One of the key reasons for developing DS2 was to provide parallel processing capability to SAS. Parallel execution is accomplished by executing a program/process in DS2 on either a single machine running multiple threads or to a distributed computing environment if SAS GRID is licensed. Threads refer to the tasks a CPU core can process simultaneously. For instance, many modern PCs have at least 2 cores, and some have 4. Today each of these cores can run up to 2 threads.

When the SAS process is CPU bound the SAS log will indicate REAL time (CPU time is essentially the same) as Running time. Multiple threads in this case can decrease CPU time. If, however, the process is I/O bound, and REAL time is greater than CPU time, then no amount of additional threads will improve efficiency. In fact, adding more threads may decrease overall performance.

The next example was taken from Chapter 6, Example 6.3.1 in Jordan (2016) and modified. The next example illustrates a modified version of the SASHELP.CARS data set and adds some computations to increase processing time using parallel processing. The number of observations was increased 1000 fold to provide a better test for threading. This ran on a Windows 8.1 PC with 4 cores and we specified 4 threads to spread around the processing. The log indicates that all 4 threads were utilized, but thread number 3 received the majority of the action. In general, parallel processing of data programs in DS2 can dramatically reduce the required CPU time if the application is not I/O bound.

Parallel Processing Code:

```
data cars(drop=i) ;
  set sashelp.cars(keep=make model type) ;
  do i = 1 to 1000 ;
     output;
  end ;
run ;

proc ds2 ;
 thread work.myThread(double flag)/overwrite=y ;
   dcl int ThreadNo ;
   dcl int Count ;
   method run() ;
```

```
      set work.cars ;
      count+1 ;
   end ;
   method term() ;
      ThreadNo=_threadid_ ;
      put ThreadNo= Count= ;
   end ;
endthread ;
run ;
quit ;

proc ds2 ;
data cars3 / overwrite=yes ;
   dcl double i j k ;
   dcl thread work.myThread t ;
   method init() ;
      t.setparms(1) ;
   end ;
   method run() ;
      set from t threads=4 ;
      do i = 1 to 100 ;
        do j = 1 to 50 ;
          k = (j*i)**3 ;
        end ;
      end ;
   end ;
enddata ;
run ;
quit ;
```

Parallel Processing Log:

```
62         data cars(drop=i);
63         set sashelp.cars(keep=make model type);
64         do i = 1 to 1000;
65         output;
66         end;
67         run;

NOTE: There were 428 observations read from the data set SASHELP.CARS.
NOTE: The data set WORK.CARS has 428000 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.06 seconds
      cpu time            0.06 seconds

68
69         Proc ds2;
70         thread work.myThread(double flag)/overwrite=y;
71            dcl int ThreadNo;
72            dcl int Count;
73            method run();
74               set work.cars;
75               count+1;
76            end;
77            method term();
78               ThreadNo=_threadid_;
79               put ThreadNo= Count=;
80            end;
81         endthread;
82         run;
NOTE: Created thread mythread in data set work.mythread.
NOTE: Execution succeeded. No rows affected.
83         quit;
```

```
NOTE: PROCEDURE DS2 used (Total process time):
      real time           0.05 seconds
      cpu time            0.04 seconds
84
85         proc ds2;
86         data cars3 / overwrite=yes;
87            dcl double i j k;
88            dcl thread work.myThread t;
89            method init();
90               t.setparms(1);
91            end;
92            method run();
93               set from t threads=4;
94               do i = 1 to 100;
95                 do j = 1 to 50;
96                 k = (j*i)**3;
97                 end;
98               end;
99            end;
100        enddata;
101        run;
ThreadNo=3 Count=220640
ThreadNo=4 Count=120240
ThreadNo=1 Count=25440
ThreadNo=2 Count=61680
NOTE: Execution succeeded. 428000 rows affected.
102        quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time           19.59 seconds
      cpu time            20.80 seconds
```

SAS Grid

Introduced in SAS 9.1.3, the separately licensed SAS Grid Manager adds to the parallel processing capabilities of SAS/CONNECT and many other capabilities required by large data, enterprise grid deployments. It provides efficient resource allocation, load balancing and prioritization for SAS solutions running in a shared grid environment. It also separates the SAS applications from the infrastructure executing the applications. This allows hardware resources to grow or shrink as needed, and provides tolerance for hardware failures in the grid system. The SAS Grid Manager can modernize your existing SAS environment.

- Multiple users - SAS® Enterprise Guide users can submit their SAS programs and tasks to a SAS grid for execution.
- Parallel workloads – Some SAS applications consists of sub-tasks that are independent units of work and can be distributed across a grid and executed in parallel.
- Enterprise scheduling - Scheduling production jobs is an important function in just about every enterprise. SAS provides the Schedule Manager plug-in with SAS® Management Console for creating SAS workflows and schedule them.

SAS Grid Manager integrates the resource management and scheduling capabilities of the Platform Suite for SAS with the SAS 4GL syntax, and subsequently with several SAS products, like Enterprise Miner, SAS® Forecast Server, SAS/STAT, SAS/OR and SAS/ETS. This enables you to create a managed, flexible, and shared environment to efficiently process and analyze data. According to Galati (2008), “SAS/OR takes advantage of SAS Grid functionality in solving some of the following problems:"

- Monte Carlo methods (Fishman 1995)
- Decomposition methods such as Dantzig-Wolfe, Benders, Lagrangian relaxation (Galati and Ralphs 2005)
- Multi-start methods for global optimization (Horst and Tuy 1993)
- Genetic and evolutionary algorithms (Levine 1994)
- Simulation (Robinson 2004)
- Location problems (Pierre, et al 2007)

With the SAS In-Database Code Accelerator processing of the data on the server can be transferred to high performance host database systems like HADOOP or TERADATA.

Exploring PROC SCAPROC – The SAS Code Analyzer

Rabb (2010) describes PROC SCAPROC – the SAS Code Analyzer as an effective tool for maintaining and modifying legacy applications and program code. Users are able to capture information about input, output, and macro variables from an executing SAS job with the SCAPROC procedure. In this next example, a file is created with the RECORD statement and the content of the SAS Code Analyzer is written to the file with the WRITE statement.

PROC SCAPROC Code:

```
proc scaproc ;
  record '/folders/myfolders/PROC SCAPROC with Record and Write.txt' ;
run ;

proc sort data=sashelp.Cars(keep=Origin Type Make MSRP)
          out=work.Cars_Sorted ;
  where MSRP < 20000 ;
  by Origin Type Make MSRP ;
run ;

ods HTML path="/folders/myfolders"  (url=none)
         file="PROC-PRINT-with-Style.html"
         style=HTMLBlue ;
proc print data=work.Cars_Sorted
  style (data)  = [background=Blue foreground=white]
  style (obs)   = [background=red foreground=white]
  style (total) = [background=yellow foreground=black] ;
  by Origin Type ;
  id Origin Type Make ;
  format msrp dollar12.0 ;
  sum MSRP ;
run ;
ods HTML close ;

proc scaproc ;
  write ;
run ;
```

PROC SCAPROC Results:

```
/* JOBSPLIT: JOBSTARTTIME 09JUL2017:07:10:42.53 */
/* JOBSPLIT: TASKSTARTTIME 09JUL2017:07:10:42.53 */
/* JOBSPLIT: DATASET INPUT SEQ #C00003.CARS.DATA */
/* JOBSPLIT: LIBNAME #C00003 V9 '/opt/sasinside/SASHome/SASFoundation/9.4/sashelp' */
/* JOBSPLIT: CONCATMEM #C00003 SASHELP */
/* JOBSPLIT: LIBNAME SASHELP V9 '(
'/opt/sasinside/SASHome/SASFoundation/9.4/nls/u8/sascfg'
'/opt/sasinside/SASHome/SASFoundation/9.4/nls/u8/sashelp'
'/opt/sasinside/SASHome/SASFoundation/9.4/nls/en/sascfg'
'/opt/sasinside/SASHome/SASFoundation/9.4/sashelp' )' */
/* JOBSPLIT: DATASET OUTPUT SEQ WORK.CARS_SORTED.DATA */
```

```
/* JOBSPLIT: LIBNAME WORK V9
'/tmp/SAS_workEF5700002373_localhost.localdomain/SAS_work98EC00002373_localhost.locald
omain' */
/* JOBSPLIT: DATASET OUTPUT SEQ WORK.SORTTMP000000000000000000012.DATA */
/* JOBSPLIT: LIBNAME WORK V9
'/tmp/SAS_workEF5700002373_localhost.localdomain/SAS_work98EC00002373_localhost.locald
omain' */
/* JOBSPLIT: ITEMSTORE INPUT WORK.TEMPLAT */
/* JOBSPLIT: ITEMSTORE INPUT SASUSER.TEMPLAT */
/* JOBSPLIT: FILE OUTPUT /folders/myfolders/PROC-PRINT-with-Style.html */
/* JOBSPLIT: SYMBOL GET SYSSORTDETAILS */
/* JOBSPLIT: SYMBOL GET SYSSORTTRACELEVEL */
/* JOBSPLIT: SYMBOL GET SYSSORTTRACE */
/* JOBSPLIT: ELAPSED 222  */
/* JOBSPLIT: SYSSCP LIN X64 */
/* JOBSPLIT: PROCNAME SORT */
/* JOBSPLIT: STEP SOURCE FOLLOWS */

proc sort data=sashelp.Cars(keep=Origin Type Make MSRP)
           out=work.Cars_Sorted ;
  where MSRP < 20000 ;
  by Origin Type Make MSRP ;
run ;
ods HTML path="/folders/myfolders"
         (url=none)
         file="PROC-PRINT-with-Style.html"
         style=HTMLBlue ;

/* JOBSPLIT: TASKSTARTTIME 09JUL2017:07:10:42.75 */
/* JOBSPLIT: DATASET INPUT MULTI WORK.CARS_SORTED.DATA */
/* JOBSPLIT: LIBNAME WORK V9
'/tmp/SAS_workEF5700002373_localhost.localdomain/SAS_work98EC00002373_localhost.locald
omain' */
/* JOBSPLIT: CATALOG INPUT #C00003.COREPRN.AFM_COM____1.PSL */
/* JOBSPLIT: LIBNAME #C00003 V9 '/opt/sasinside/SASHome/SASFoundation/9.4/sashelp' */
/* JOBSPLIT: CONCATMEM #C00003 SASHELP */
/* JOBSPLIT: LIBNAME SASHELP V9 '(
'/opt/sasinside/SASHome/SASFoundation/9.4/nls/u8/sascfg'
'/opt/sasinside/SASHome/SASFoundation/9.4/nls/u8/sashelp'
'/opt/sasinside/SASHome/SASFoundation/9.4/nls/en/sascfg'
'/opt/sasinside/SASHome/SASFoundation/9.4/sashelp' )' */
/* JOBSPLIT: CATALOG INPUT #C00003.COREPRN.AFM_TIB____1.PSL */
/* JOBSPLIT: LIBNAME #C00003 V9 '/opt/sasinside/SASHome/SASFoundation/9.4/sashelp' */
/* JOBSPLIT: CONCATMEM #C00003 SASHELP */
/* JOBSPLIT: LIBNAME SASHELP V9 '(
'/opt/sasinside/SASHome/SASFoundation/9.4/nls/u8/sascfg'
'/opt/sasinside/SASHome/SASFoundation/9.4/nls/u8/sashelp'
'/opt/sasinside/SASHome/SASFoundation/9.4/nls/en/sascfg'
'/opt/sasinside/SASHome/SASFoundation/9.4/sashelp' )' */
/* JOBSPLIT: FILE INPUT /opt/sasinside/SASHome/ReportFontsforClients/9.4/saswalb.ttf
*/
/* JOBSPLIT: FILE INPUT /opt/sasinside/SASHome/ReportFontsforClients/9.4/saswcur.ttf
*/
/* JOBSPLIT: ITEMSTORE UPDATE Work.SASTMP-000000013 */
/* JOBSPLIT: FILE OUTPUT /folders/myfolders/PROC SCAPROC with Record and Write.txt */
/* JOBSPLIT: ELAPSED 277  */
/* JOBSPLIT: PROCNAME PRINT */
/* JOBSPLIT: STEP SOURCE FOLLOWS */
proc print data=work.Cars_Sorted
  style (data)  = [background=Blue foreground=white]
  style (obs)   = [background=red foreground=white]
  style (total) = [background=yellow foreground=black] ;
  by Origin Type ;
  id Origin Type Make ;
  format msrp dollar12.0 ;
```

```
  sum MSRP ;
run ;
ods HTML close ;

/* JOBSPLIT: JOBENDTIME 09JUL2017:07:10:43.03 */
/* JOBSPLIT: END */
```

Conclusion

Large inventories of SAS applications and program code have been developed and supported by organizations since the mid-1970s. In many organizations, novice and experienced SAS® programmers have been tasked with the responsibility to support their organization's legacy applications, programs and code. This paper explores many tips, techniques and examples to help guide the modernization of applications and program code well into the 21st century and beyond. The recommended techniques and approaches are designed to provide a foundation of things to consider during the modernizing process. We also provide guidance on the use of the SCAPROC procedure – the SAS code analyzer – to analyze metadata about the contents of SAS code, and to streamline, scale and modernize code constructs, algorithms, functions, and legacy application program code.

References

DS2 References and Suggested Reading

Barnes, Arila; Jared Peterson; Saratendu Sethi (2013). *"Unleashing the Power of Unified Text Analytics to Categorize Call Center Data,"* Proceedings of the 2013 SAS Global Forum (SGF) Conference.

Brooks, Chris (2016). *"Tips and Techniques for User-Defined Packages in SAS® DS2,"* Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Eberhardt, Peter and Xue Yao (2015). *"DS2 with Both Hands on the Wheel,"* Proceedings of the 2015 SAS Global Forum (SGF) Conference.

Eberhardt, Peter and Xue Yao (2014). *"I Object: SAS® Does Objects with DS2,"* Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Jordan, Mark (2016). *"Mastering the SAS® DS2 Procedure: Advanced Data Wrangling Techniques,"* SAS Press, SAS Institute, Cary, NC, USA.

.Jordan, Mark L (2014). *"Using Base SAS® to Extend the SAS® System,"* SAS® Institute Inc, Cary NC, Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Kaufmann, Shaun (2016). *"High-Performance Data Access with FedSQL and DS2,"* Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Kaufmann, Shaun (2014). *"A Paradigm Shift: Complex Data Manipulations with DS2 and In-Memory Data Structures,"* Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Kumbhakarna, Viraj R., (2017). *"PROC DS2: What's in it for You?,"* Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Lal, Rajesh, Experis (2014). *"DS2: The New and Improved DATA Step in SAS®,"* Proceedings of the 2014 MidWest SAS Users Group (MWSUG) Conference.

Massey, J. Gregory; Radhikha Myneni, M.; Adrian Mattocks; and Eric C. Brinsfield (2014). *"Extracting Key Concepts from Unstructured Medical Reports Using SAS® Text Analytics and SAS® Visual Analytics,"* Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Matsey, Bob and Tho Nguyen (2015), *"The Power of DS2 Programming,"* Proceedings of the 2015 SouthEast SAS Users Group (SESUG) Conference.

SAS Institute Inc. (2016). *SAS® 9.4 DS2 Language Reference, Sixth Edition*. Cary, NC

Secosky, Jason; Robert Ray; and Greg Otto (2014). *"Parallel Data Preparation with the DS2 Programming Language,"*, Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Efficiency and Performance Tuning References and Suggested Reading

Brown, Tony and Margaret Crevar (2016). "*Architecting Your SAS Grid®: Networking for Performance*," Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Cohen, Robert A. and Robert N. Rodriguez (2013). "*High-Performance Statistical Modeling*," Proceedings of the 2013 SAS Global Forum (SGF) Conference.

Kaufmann, Shaun (2016). *"High-Performance Data Access with FedSQL and DS2,"* Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Lafler, Kirk Paul (2016). "*Top Ten SAS® Performance Tuning Techniques*," Proceedings of the 2016 MidWest SAS Users Group (MWSUG) Conference.

Lavery, Russ (2013). "*Fast Access Tricks for Large Sorted SAS Files*," Proceedings of the 2013 MidWest SAS Users Group (MWSUG) Conference.

Lui, Lingqun (2017). "*SAS Advanced Programming with Efficiency in Mind: A Real Case Study*," Proceedings of the 2017 Michigan SAS Users Group (MISUG) Conference.

Warner-Freeman, Jennifer K. (2007). "*I Cut My Processing Time By 90% Using Hash Tables - You Can Do It Too!*," Proceedings of the 2007 North East SAS Users Group (NESUG) Conference.

Williams, Michael; Gretel Easter and Steve Bradsher (2009). *"Troubleshoot Your Performance Issues: SAS® Technical Support Shows You How,"* Proceedings of the 2009 SAS Global Forum (SGF) Conference.

General References and Suggested Reading

Ford, Andrew P.; Troy B. Wolfe; and Shiva Srinivasan (2008). *"Reinvent Legacy Software with SAS®, the Web, and OLAP Reporting,"* Proceedings of the 2008 SAS Global Forum (SGF) Conference.

Green, Adam (2017). "*5 Signs You Need to Modernize a Legacy Application*," BPlans, owned and operated by Palo Alto Software.

Jordan, Mark L. (2014). *"Using Base SAS® to Extend the SAS® System,"* SAS® Institute Inc, Cary NC, Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Techopedia.com (June 30th, 2017). *Legacy Code Definition from Techopedia.com*.

Hash Object References and Suggested Reading

Burlew, Michele M. (2012), "*SAS® Hash Object Programming Made Easy*," SAS Press, SAS Institute, Cary, NC, USA.

Dorfman, Paul M. and Don Henderson (2017). "*Beyond Table Lookup: The Versatile SAS® Hash Object*," Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Dorfman, Paul M. (2016). "*Using the SAS® Hash Object with Duplicate Key Entries*," Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Dorfman, Paul and Peter Eberhardt (2010). "*Two Guys on Hash*," Proceedings of the 2010 South East SAS Users Group (SESUG) Conference.

Dorfman, Paul (2009). "*The SAS® Hash Object in Action*," Proceedings of the 2009 South East SAS Users Group (SESUG) Conference.

Lafler, Kirk Paul (2016). "*An Introduction to SAS® Hash Programming Techniques*," Proceedings of the 2016 SouthEast SAS Users Group (SESUG) Conference.

Loren, Judy (2008). "*How Do I Love Hash Tables? Let Me Count The Ways!*," Proceedings of the 2008 SAS Global Forum (SGF) Conference.

Mazloom, Dari (2017). "*SAS Hash Objects, Demystified*," Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Sakya, Daniel (2012). "*SAS® HASH Programming Basics*," Proceedings of the 2012 South Central SAS Users Group (SCSUG) Conference.

Schacherer, Chris (2015). "*Introduction to SAS® Hash Objects*," Proceedings of the 2015 SAS Global Forum (SGF) Conference.

Secosky, Jason and Janice Bloom (2007). "*Getting Started with the DATA Step Hash Object*," Proceedings of the 2007 SAS Global Forum (SGF) Conference.

Warner-Freeman, Jennifer K. (2007). "*I Cut My Processing Time By 90% Using Hash Tables - You Can Do It Too!*," Proceedings of the 2007 North East SAS Users Group (NESUG) Conference.

Macro References and Suggested Reading

Carpenter, Art (2016). *Carpenter's Complete Guide to the SAS® Macro Language, Third Edition*, SAS Institute Inc., Cary, NC.

Lui, Lingqun (2007). "*Passing Data Set Values into Application Parameters*," Proceedings of the 2007 MidWest SAS Users Group (MWSUG) Conference.

Roberts, Clark (1997). *"Building and Using Macro Variable Lists,"* Proceedings of the 1997 SAS Users Group International (SUGI) Conference.

Operations Research (OR) References and Suggested Reading

Fishman, G. (1995), *Monte Carlo: Concepts, Algorithms, and Applications*, New York: Springer Verlag.

Galati, M.V. and Ted K. Ralphs (2005), "*Decomposition in Integer Linear Programming*," in J. Karlof, ed., Integer Programming: Theory and Practice, The Pennsylvania State University: CRC Press.

Horst, R. and Tuy, H. (1993), *Global Optimization: Deterministic Approaches*, New York: Springer-Verlag.

Levine, D. (1994), *A Parallel Genetic Algorithm for the Set Partitioning Problem*, Ph.D. thesis, Illinois Institute of Technology, Chicago, IL.

Robinson, Stewart (2004). *Simulation—The Practice of Model Development and Use*, John Wiley & Sons.

Hansen, Pierre, Nenad ML ADENOVIC, and Eric Taillard (2007), "*Location Problems - Heuristic Solution of the Multisource Weber Problem as a p-Median Problem*," (February 4, 2008).

Wicklin, Rick (2016). "*Solve linear programming problems in SAS*," blogs.sas.com.

PROC FCMP References and Suggested Reading

Adams, John H. (2010). "*The new SAS 9.2 FCMP Procedure, what functions are in your future?*," Proceedings of the 2010 PharmaSUG Conference.

Carpenter, Art. (2013). "*Using PROC FCMP to the Fullest: Getting Started and Doing More*", Proceedings of 2013 SAS Global Forum (SGF) Conference.

Deguire, Yves, Xiyun (Cheryl) Wang (2013). "*Using SAS® PROC FCMP in SAS® System Development - Real Examples*," Proceedings of the 2010 SAS Global Forum (SGF) Conference.

Eberhardt, Peter (2010). "*Functioning at an Advanced Level: PROC FCMP and PROC PROTO*," Proceedings of the 2010 SAS Global Forum (SGF) Conference.

Eberhardt Peter (2009). "*A Cup of Coffee and Proc FCMP: I Cannot Function Without Them*," Proceedings of the 2009 SAS Global Forum (SGF) Conference.

Eckler, Lisa (2013). "*FCMP – Why?*," Proceedings of the 2013 SAS Global Forum (SGF) Conference.

Henrick, Andrew, Mike Whitcher, and Karen Croft (2017). "*Dictionaries: Referencing a New PROC FCMP Data Type*," Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Henrick, Andrew, Donald Erdman, and Karen Croft (2015). "*Helping You C What You Can Do with SAS®*," Proceedings of the 2015 SAS Global Forum (SGF) Conference.

Henrick, Andrew, Donald Erdman, and Stacey Christian (2013). "*Hashing in PROC FCMP to Enhance Your Productivity*," Proceedings of the 2013 SAS Global Forum (SGF) Conference.

Rithy, Danny (2015). "*Getting Started with PROC FCMP*," Proceedings of the 2015 Western Users of SAS Software (WUSS) Conference.

Secosky, Jason (2012). "*Executing a PROC from a DATA Step*," Proceedings of the 2012 SAS Global Forum (SGF) Conference.

Secosky, Jason (2007). "*User-Written DATA Step Functions*," Proceedings of the 2007 SAS Global Forum (SGF) Conference.

PROC SCAPROC References and Suggested Reading

Rabb, Merry (2010). *"Thoroughly Modern SAS®: The SAS Code Analyzer Helps Bring Programs Up to Date,"* Proceedings of the 2010 SAS Global Forum (SGF) Conference.

Thies, Eric and Rick Langston (2008). *"Introducing the SAS® Code Analyzer,"* Proceedings of the 2008 SAS Global Forum (SGF) Conference.

SAS Grid References and Suggested Reading

Bosso, Marlos A. (2016). "*Creating a Strong Business Case for SAS® Grid Manager: Translating Grid Computing Benefits to Business Benefits*," Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Brown, Tony and Margaret Crevar (2016). "*Architecting Your SAS Grid®: Networking for Performance*," Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Cohen, Robert A. and Robert N. Rodriguez (2013). "*High-Performance Statistical Modeling*," Proceedings of the 2013 SAS Global Forum (SGF) Conference.

Doninger, Cheryl, Zhiyong Li, and Bryan Wolfe (2014). "*Best Practices for Implementing High Availability for SAS® 9.4*," Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Doninger, Cheryl and Glenn Horton (2008). "*SAS® Grid 101: How It Can Modernize Your Existing SAS® Environment*," Proceedings of the 2008 SAS Global Forum (SGF) Conference.

Galati, Matthew, Doug Haigh, Rob Pratt, and Ivan Oliveira (2008). "*Using SAS/OR® and SAS® Grid Manager to Solve Optimization Problems on the Grid*," Proceedings of the 2008 SAS Global Forum (SGF) Conference.

Leonard, Michael, Cheryl Doninger, and Udo Sglavo (2014). "*High-Performance Forecasting Using SAS® Grid Manager*," Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Nitschinger, Manuel and Phillip Manschek (2014). "*SAS® Grid – What They Didn't Tell You*," Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Oliveira, Ivan, Rob Pratt, and Charles Dulaney (2009). "*Using the SAS/OR® OPTMODEL Procedure to Assign Students to Schools in the Wake County Public School System*," Proceedings of the 2008 SAS Global Forum (SGF) Conference.

SAS Institute Inc., *SAS/CONNECT® 9.4 User's Guide, Fourth Edition*., Cary NC.

SAS Institute Inc., *SAS/GRID®, Grid Computing in SAS® 9.4, Fifth Edition*., Cary NC.

SAS Programming Techniques References and Suggested Reading

Benjamin, William E. Jr. (2012). *"Leave Your Bad Code Behind: 50 Ways to Make Your SAS® Code Execute More Efficiently,"* Proceedings of the 2012 SAS Global Forum (SGF) Conference.

Cassidy, Deb (2003). "*Keeping Up With the FUN: New Functions in SAS 9*," Proceedings of the 2003 SouthEast SAS Users Group Conference.

Cody, Ron (2012). "*A Survey of Some of the Most Useful SAS® Functions*," Proceedings of the 2012 SAS Global Forum (SGF) Conference.

Gupta, Sunil (2006). "*WHERE vs. IF Statements: Knowing the Difference in How and When to Apply*," Proceedings of the 2006 SAS Users Group International (SUGI) Conference.

Hecht, Darylene (2011). "*PROC PRINT and ODS: Teaching an Old PROC New Tricks*," Proceedings of the 2011 SAS Global Forum (SGF) Conference.

Horstman, Joshua M. (2017). "*Beyond IF THEN ELSE: Techniques for Conditional Execution of SAS® Code*," Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Lafler, Kirk Paul (2019). *PROC SQL: Beyond the Basics Using SAS, Third Edition*, SAS Institute Inc., Cary, NC, USA.

Lafler, Kirk Paul (2017). "*Best Practice Programming Techniques for SAS® Users*," Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Lafler, Kirk Paul (2017). "*Removing Duplicates Using SAS®*," Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Lafler, Kirk Paul (2014). "*Conditional Processing Using the Case Expression in PROC SQL*," Proceedings of the 2014 South Central SAS Users Group (SCSUG) Conference.

Lafler, Kirk Paul (2013). *PROC SQL: Beyond the Basics Using SAS, Second Edition*, SAS Institute Inc., Cary, NC, USA.

Lafler, Kirk Paul (2009). "*SAS® Macro Programming Tips and Techniques*," Proceedings of the 20009 SAS Global Forum (SGF) Conference.

Lavery, Russ (2016). "*An Animated Guide: The Internals of PROC REPORT*," Proceedings of the 2016 MidWest SAS Users Group (MWSUG) Conference.

Lui, Lingqun (2007). "*Passing Data Set Values into Application Parameters*," Proceedings of the 2007 MidWest SAS Users Group (MWSUG) Conference.

Repole Jr, Warren (2009). *"Don't Be a SAS® Dinosaur: Modernizing Programs with Base SAS 9.2 Enhancements,"* Proceedings of the 2009 SAS Global Forum (SGF) Conference.

Riba, S. David (1996). *"Redesigning a Legacy: Techniques of a Quality Partner,"* Proceedings of the 1996 SAS Users Group International (SUGI) Conference.

Roberts, Clark; Deborah Testa and Russell Holmes (19997). *"Audit Trail Plug-ins for SAS® Software Applications,"* Proceedings of the 1999 Western Users of SAS Software (WUSS) Conference.

Roberts, Clark (1997). *"Building and Using Macro Variable Lists,"* Proceedings of the 1997 SAS Users Group International (SUGI) Conference.

Shapiro, Mira (2016). "*SAS® Functions You May Have Been MISSING*," Proceedings of the 2016 PharmaSUG Conference.

Sun, GuanGhui (Brian) (2011). "*Why Dummy Variable Makes You SMART, and How to Do it SEXY*," Proceedings of the 2011 Western Users of SAS Software (WUSS) Conference.

Venam, Srinivas; Manvitha Yennam; and Phaneendhar Vanam (2016). "*Good Programming Practice [GPP] in SAS® & Clinical Trials*," Proceedings of the 2016 Western Users of SAS Software (WUSS) Conference.

Wang, Hui (2015). "*Creating Data-Driven SAS® Code with CALL EXECUTE*," Proceedings of the 2015 PharmaSUG Conference.

Whitlock, Ian (1998). "*CALL EXECUTE: How and Why*," Proceedings of the 1998 SAS Users Group International (SUGI) Conference.

Text Analytics References and Suggested Reading

Barnes, Arila; Jared Peterson; Saratendu Sethi (2013). *"Unleashing the Power of Unified Text Analytics to Categorize Call Center Data,"* Proceedings of the 2013 SAS Global Forum (SGF) Conference.

Massey, J. Gregory; Radhikha Myneni, M.; Adrian Mattocks; and Eric C. Brinsfield (2014). *"Extracting Key Concepts from Unstructured Medical Reports Using SAS® Text Analytics and SAS® Visual Analytics,"* Proceedings of the 2014 SAS Global Forum (SGF) Conference.

WIN32API References and Suggested Reading

Appleman, Dan (1999), *Dan Appleman's Visual Basic Programmer's Guide to the Win32 API*, Sams.

Cody, Ron, (2010). *SAS Functions by Example, Second Edition*, SAS Press, Cary, NC., SAS On-line Doc., SAS Institute, Cary, NC.

Cody, Ron (2012). "*A Survey of Some of the Most Useful SAS® Functions*," Proceedings of the 2012 SAS Global Forum (SGF) Conference.

DeVenezia ,Richard A., and Judy Loren (2008). "*Using CALL MODULE in SAS® on Linux, or I get by with a little help from my friends*," Proceedings of the 2008 SAS Global Forum (SGF) Conference.

Foster, Edward (2006). "*Using the WIN32 API from SAS*," PhUSE 2006.

Lal, Rajesh (2014). "*Using Microsoft Windows DLLs within SAS® Programs*," Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Langston Rick (2015). "*When I'm 64-bit: How to Still Use 32-bit DLLs in Microsoft Windows*," Proceedings of the 2015 SAS Global Forum (SGF) Conference.

Langston, Richard D. (1995). "*Examples Using The MODULE Routines In PC Environments*," Proceedings of the 1995 SAS Users Group International Conference.

Microsoft Office Site (2010), *Microsoft Office 2010 Win32API_PtrSafe for Visual Basic* to download the Win32API_PtrSafe.txt file and related information, then install it).

Microsoft Corporation (1994-1999). *Win32 API Declarations for Visual Basic (win32api.txt)*.

Microsoft Corporation, *MDSN: Develop desktop applications and drivers*.

SAS Institute, Inc., *SAS® 9.4 Companion for Windows, Fifth Edition*.

Acknowledgments

The authors thank the SESUG 2023 Conference Committee, particularly the Development and Support Section Chairs, Bobbie Frye and Jinson Erinjeri, for accepting our abstract and paper; the SESUG 2023 Academic Chair, Mel Alexander, and the Operation Chair, Kelly Smith, for organizing and supporting a great "live" conference event; SAS Institute Inc. for providing SAS users with wonderful software; and SAS users everywhere for being the nicest people anywhere!

Trademark Citations

Data Sets Used in Examples

The examples presented in this paper include the RUGs_2015 and RUGs_2016 data sets; and several in the SASHELP library including the CARS, HEART, SHOES, and STOCKS data sets You'll be able to use these data sets for example purposes and for testing the enclosed code examples.

The RUGs_2015 data set consists of 4 observations and 3 variables, illustrated below.

	RUG	Number_Papers	Year
1	MWSUG	96	2015
2	SCSUG	29	2015
3	SESUG	148	2015
4	WUSS	102	2015

Data Set #1. RUGs_2015

The RUGs_2016 data set consists of 4 observations and 3 variables, illustrated below.

	RUG	Number_Papers	Year
1	MWSUG	124	2016
2	SCSUG	62	2016
3	SESUG	148	2016
4	WUSS	112	2016

Data Set #2. RUGs_2016

The MOVIES data set consists of 22 observations and 6 variables, illustrated below.

	Title	Length	Category	Year	Studio	Rating
1	Brave Heart	177	Action Adventure	1995	Paramount Pictures	R
2	Casablanca	103	Drama	1942	MGM / UA	PG
3	Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13
4	Coming to America	116	Comedy	1988	Paramount Pictures	R
5	Dracula	130	Horror	1993	Columbia TriStar	R
6	Dressed to Kill	105	Drama Mysteries	1980	Filmways Pictures	R
7	Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13
8	Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13
9	Jaws	125	Action Adventure	1975	Universal Studios	PG
10	Jurassic Park	127	Action	1993	Universal Pictures	PG-13
11	Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R
12	Michael	106	Drama	1997	Warner Brothers	PG-13
13	National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13
14	Poltergeist	115	Horror	1982	MGM / UA	PG
15	Rocky	120	Action Adventure	1976	MGM / UA	PG
16	Scarface	170	Action Cops & Robber	1983	Universal Studios	R
17	Silence of the Lambs	118	Drama Suspense	1991	Orion	R
18	Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	PG
19	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG
20	The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R
21	The Wizard of Oz	101	Adventure	1939	MGM / UA	G
22	Titanic	194	Drama Romance	1997	Paramount Pictures	PG-13

Data Set #3. MOVIES

The ACTORS data set consists of 13 observations and 3 variables, illustrated below.

	Title	Actor_Leading	Actor_Supporting
1	Brave Heart	Mel Gibson	Sophie Marceau
2	Christmas Vacation	Chevy Chase	Beverly D'Angelo
3	Coming to America	Eddie Murphy	Arsenio Hall
4	Forrest Gump	Tom Hanks	Sally Field
5	Ghost	Patrick Swayze	Demi Moore
6	Lethal Weapon	Mel Gibson	Danny Glover
7	Michael	John Travolta	Andie MacDowell
8	National Lampoon's Vacation	Chevy Chase	Beverly D'Angelo
9	Rocky	Sylvester Stallone	Talia Shire
10	Silence of the Lambs	Anthony Hopkins	Jodie Foster
11	The Hunt for Red October	Sean Connery	Alec Baldwin
12	The Terminator	Arnold Schwarzenegge	Michael Biehn
13	Titanic	Leonardo DiCaprio	Kate Winslet

Data Set #4. ACTORS

The SASHELP.CARS data set consists of 428 observations and 15 variables, illustrated below.

	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	265	17	23	4451	106	189
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2	4	200	24	31	2778	101	172
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	200	22	29	3230	105	183
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	270	20	28	3575	108	186
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	225	18	24	3880	115	197
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6	225	18	24	3893	115	197
7	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2	6	290	17	24	3153	100	174
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8	4	170	22	31	3252	104	179
9	Audi	A41.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8	4	170	23	30	3638	105	180
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3	6	220	20	28	3462	104	179
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	All	$33,430	$30,366	3	6	220	17	26	3583	104	179
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	All	$34,480	$31,388	3	6	220	18	25	3627	104	179
13	Audi	A6 3.0 4dr	Sedan	Europe	Front	$36,640	$33,129	3	6	220	20	27	3561	109	192
14	Audi	A6 3.0 Quattro 4dr	Sedan	Europe	All	$39,640	$35,992	3	6	220	18	25	3880	109	192
15	Audi	A4 3.0 convertible 2dr	Sedan	Europe	Front	$42,490	$38,325	3	6	220	20	27	3814	105	180
16	Audi	A4 3.0 Quattro convertible 2dr	Sedan	Europe	All	$44,240	$40,075	3	6	220	18	25	4013	105	180
17	Audi	A6 2.7 Turbo Quattro 4dr	Sedan	Europe	All	$42,840	$38,840	2.7	6	250	18	25	3836	109	192
18	Audi	A6 4.2 Quattro 4dr	Sedan	Europe	All	$49,690	$44,936	4.2	8	300	17	24	4024	109	193
19	Audi	A8 L Quattro 4dr	Sedan	Europe	All	$69,190	$64,740	4.2	8	330	17	24	4399	121	204
20	Audi	S4 Quattro 4dr	Sedan	Europe	All	$48,040	$43,556	4.2	8	340	14	20	3825	104	179
21	Audi	RS 6 4dr	Sports	Europe	Front	$84,600	$76,417	4.2	8	450	15	22	4024	109	191
22	Audi	TT 1.8 convertible 2dr (coupe)	Sports	Europe	Front	$35,940	$32,512	1.8	4	180	20	28	3131	95	159
23	Audi	TT 1.8 Quattro 2dr (convertible)	Sports	Europe	All	$37,390	$33,891	1.8	4	225	20	28	2921	96	159
24	Audi	TT 3.2 coupe 2dr (convertible)	Sports	Europe	All	$40,590	$36,739	3.2	6	250	21	29	3351	96	159
25	Audi	A6 3.0 Avant Quattro	Wago	Europe	All	$40,840	$37,060	3	6	220	18	25	4035	109	192
26	Audi	S4 Avant Quattro	Wago	Europe	All	$49,090	$44,446	4.2	8	340	15	21	3936	104	179
27	BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3	6	225	16	23	4023	110	180
28	BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4	8	325	16	22	4824	111	184

Data Set #5. SASHELP.CARS

The SASHELP.HEART data set consists of 5,209 observations and 17 variables, illustrated below.

	Status	DeathCause	AgeCHDdiag	Sex	AgeAtStart	Height	Weight	Diastolic	Systolic	MRW	Smoking	AgeAtDeath	Cholesterol	Chol_Status	BP_Status	Weight_Status	Smoking_Status
1	Dead	Other	.	Female	29	62.5	140	78	124	121	0	55	.		Normal	Overweight	Non-smoker
2	Dead	Cancer	.	Female	41	59.75	194	92	144	183	0	57	181	Desirable	High	Overweight	Non-smoker
3	Alive		.	Female	57	62.25	132	90	170	114	10	.	250	High	High	Overweight	Moderate (6-15)
4	Alive		.	Female	39	65.75	158	80	128	123	0	.	242	High	Normal	Overweight	Non-smoker
5	Alive		.	Male	42	66	156	76	110	116	20	.	281	High	Optimal	Overweight	Heavy (16-25)
6	Alive		.	Female	58	61.75	131	92	176	117	0	.	196	Desirable	High	Overweight	Non-smoker
7	Alive		.	Female	36	64.75	136	80	112	110	15	.	196	Desirable	Normal	Overweight	Moderate (6-15)
8	Dead	Other	.	Male	53	65.5	130	80	114	99	0	77	276	High	Normal	Normal	Non-smoker
9	Alive		.	Male	35	71	194	68	132	124	0	.	211	Borderline	Normal	Overweight	Non-smoker
10	Dead	Cerebral Vascular Disease	.	Male	52	62.5	129	78	124	106	5	82	284	High	Normal	Normal	Light (1-5)
11	Alive		.	Male	39	66.25	179	76	128	133	30	.	225	Borderline	Normal	Overweight	Very Heavy (> 25)
12	Alive		57	Male	33	64.25	151	68	108	118	0	.	221	Borderline	Optimal	Overweight	Non-smoker
13	Alive		55	Male	33	70	174	90	142	114	0	.	188	Desirable	High	Overweight	Non-smoker
14	Alive		79	Male	57	67.25	165	76	128	118	15	.	.		Normal	Overweight	Moderate (6-15)
15	Alive		66	Male	44	69	155	90	130	105	30	.	292	High	High	Normal	Very Heavy (> 25)
16	Alive		.	Female	37	64.5	134	76	120	108	10	.	196	Desirable	Normal	Normal	Moderate (6-15)
17	Alive		.	Male	40	66.25	151	72	132	112	30	.	192	Desirable	Normal	Overweight	Very Heavy (> 25)
18	Dead	Cancer	56	Male	56	67.25	122	72	120	87	15	72	194	Desirable	Normal	Underweight	Moderate (6-15)
19	Alive		.	Female	42	67.75	162	96	138	119	1	.	200	Borderline	High	Overweight	Light (1-5)
20	Dead	Coronary Heart Disease	74	Male	46	66.5	157	84	142	116	30	76	233	Borderline	High	Overweight	Very Heavy (> 25)
21	Alive		.	Female	37	66.25	148	78	110	112	15	.	192	Desirable	Optimal	Overweight	Moderate (6-15)
22	Alive		.	Female	45	64	147	74	120	119	5	.	209	Borderline	Normal	Overweight	Light (1-5)
23	Alive		.	Female	59	65.75	156	74	156	122	0	.	200	Borderline	High	Overweight	Non-smoker
24	Alive		.	Female	36	63.75	122	84	132	102	0	.	184	Desirable	Normal	Normal	Non-smoker
25	Alive		.	Female	50	67.5	185	88	150	136	15	.	228	Borderline	High	Overweight	Moderate (6-15)

Data Set #6. SASHELP.HEART

The SASHELP.SHOES data set consists of 395 observations and 7 variables, illustrated below.

	Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
1	Africa	Boot	Addis Ababa	12	$29,761	$191,821	$769
2	Africa	Men's Casual	Addis Ababa	4	$67,242	$118,036	$2,284
3	Africa	Men's Dress	Addis Ababa	7	$76,793	$136,273	$2,433
4	Africa	Sandal	Addis Ababa	10	$62,819	$204,284	$1,861
5	Africa	Slipper	Addis Ababa	14	$68,641	$279,795	$1,771
6	Africa	Sport Shoe	Addis Ababa	4	$1,690	$16,634	$79
7	Africa	Women's Casual	Addis Ababa	2	$51,541	$98,641	$940
8	Africa	Women's Dress	Addis Ababa	12	$108,942	$311,017	$3,233
9	Africa	Boot	Algiers	21	$21,297	$73,737	$710
10	Africa	Men's Casual	Algiers	4	$63,206	$100,982	$2,221
11	Africa	Men's Dress	Algiers	13	$123,743	$428,575	$3,621
12	Africa	Sandal	Algiers	25	$29,198	$84,447	$1,530
13	Africa	Slipper	Algiers	17	$64,891	$248,198	$1,823
14	Africa	Sport Shoe	Algiers	9	$2,617	$9,372	$168
15	Africa	Women's Dress	Algiers	12	$90,648	$266,805	$2,690
16	Africa	Boot	Cairo	20	$4,846	$18,965	$229
17	Africa	Men's Casual	Cairo	25	$360,209	$1,063,251	$9,424
18	Africa	Men's Dress	Cairo	5	$4,051	$45,962	$97
19	Africa	Sandal	Cairo	9	$10,532	$50,430	$598
20	Africa	Slipper	Cairo	9	$13,732	$54,117	$1,216
21	Africa	Sport Shoe	Cairo	3	$2,259	$20,815	$44
22	Africa	Women's Casual	Cairo	14	$328,474	$940,851	$10,124
23	Africa	Women's Dress	Cairo	3	$14,095	$51,145	$745
24	Africa	Boot	Johannesburg	14	$8,365	$33,011	$483
25	Africa	Sandal	Johannesburg	13	$17,337	$63,003	$809
26	Africa	Slipper	Johannesburg	12	$39,452	$130,025	$1,565
27	Africa	Sport Shoe	Johannesburg	8	$5,172	$29,368	$139
28	Africa	Women's Dress	Johannesburg	4	$42,682	$120,127	$966

Data Set #7. SASHELP.SHOES

The SASHELP.STOCKS data set consists of 699 observations and 8 variables, illustrated below.

	Stock	Date	Open	High	Low	Close	Volume	AdjClose
1	IBM	01DEC05	$89.15	$89.92	$81.56	$82.20	5,976,252	$81.37
2	IBM	01NOV05	$81.85	$89.94	$80.64	$88.90	5,556,471	$88.01
3	IBM	03OCT05	$80.22	$84.60	$78.70	$81.88	7,019,666	$80.86
4	IBM	01SEP05	$80.16	$82.11	$76.93	$80.22	5,772,280	$79.22
5	IBM	01AUG05	$83.00	$84.20	$79.87	$80.62	4,801,386	$79.62
6	IBM	01JUL05	$74.30	$85.11	$74.16	$83.46	8,056,590	$82.23
7	IBM	01JUN05	$75.57	$77.73	$73.45	$74.20	6,439,536	$73.10
8	IBM	02MAY05	$76.88	$78.11	$72.50	$75.55	6,896,904	$74.43
9	IBM	01APR05	$91.49	$91.76	$71.85	$76.38	10,709,200	$75.05
10	IBM	01MAR05	$92.64	$93.73	$89.09	$91.38	5,025,627	$89.79
11	IBM	01FEB05	$93.67	$94.97	$91.55	$92.58	4,455,657	$90.97
12	IBM	03JAN05	$98.97	$99.10	$91.44	$93.42	5,960,945	$91.62
13	IBM	01DEC04	$94.50	$99.00	$94.47	$98.58	5,043,800	$96.68
14	IBM	01NOV04	$89.33	$96.63	$89.23	$94.24	5,754,876	$92.42
15	IBM	01OCT04	$85.95	$90.27	$84.29	$89.75	5,839,742	$87.85
16	IBM	01SEP04	$84.05	$87.28	$83.24	$85.74	4,719,252	$83.93
17	IBM	02AUG04	$86.87	$87.39	$81.90	$84.69	4,298,600	$82.90
18	IBM	01JUL04	$88.28	$88.44	$83.42	$87.07	5,929,023	$85.05
19	IBM	01JUN04	$88.00	$91.21	$87.30	$88.15	4,604,409	$86.10
20	IBM	03MAY04	$88.13	$89.75	$85.12	$88.59	5,395,555	$86.53
21	IBM	01APR04	$91.67	$94.55	$88.01	$88.17	5,507,214	$85.95
22	IBM	01MAR04	$96.50	$97.60	$90.28	$91.84	5,612,921	$89.53
23	IBM	02FEB04	$99.15	$100.43	$95.20	$96.50	5,392,468	$94.07
24	IBM	02JAN04	$92.86	$99.85	$89.01	$99.23	7,275,305	$96.57
25	IBM	01DEC03	$90.90	$94.12	$90.03	$92.68	5,492,295	$90.20
26	IBM	03NOV03	$89.90	$91.48	$87.72	$90.54	5,271,663	$88.12
27	IBM	01OCT03	$88.75	$94.54	$87.53	$89.48	6,999,200	$86.93
28	IBM	02SEP03	$82.40	$93.47	$82.30	$88.33	8,523,800	$85.81

Data Set #8. SASHELP.STOCKS

About the Author

Kirk Paul Lafler is an educator, developer, programmer, consultant, and data analyst; currently working as a lecturer and adjunct professor at San Diego State University and the University of California San Diego Extension; and teaching SAS, SQL, Python, Excel, and cloud-based technology courses to users around the world. Kirk has decades of programming experience and specializes in SAS software, SQL, RDBMS technologies (Oracle, SQL-Server, Teradata, DB2), Python, and other languages and productivity tools. Kirk is the author of the popular PROC SQL: Beyond the Basics Using SAS, Third Edition (SAS Press. 2019) and is actively involved with SAS, SQL, Python, R, ML, and cloud-computing user groups, conferences, and blogs as an Invited speaker, educator, keynote, and leader; and is the recipient of 27 “Best” contributed paper, hands-on workshop (HOW), and poster awards.

Clark Roberts is the principal consultant and founder of Decision Analytics and has been a SAS user since 1979. As a SAS application developer, data analyst, and programmer; Clark is the author of several published papers at SAS International, regional, special-interest, and local user group conferences; and is the recipient of a “Best” contributed paper award.

Comments and suggestions can be sent to:

Kirk Paul Lafler, sasNerd
SAS® / SQL / RDBMS / Python / Excel / Cloud-based Developer, Programmer, Consultant, Educator, Data Analyst, and Author
E-mail: KirkLafler@cs.com
LinkedIn: https://www.linkedin.com/in/KirkPaulLafler/
Twitter: @sasNerd

~ ~ ~ ~ ~

Clark Roberts
Principal SAS® Consultant and Programmer
Decision Analytics
E-mail: dacmr@hotmail.com