Paper 6-2018

Introduction to Data-driven Programming Using SAS®

Kirk Paul Lafler, Software Intelligence Corporation

Abstract

Data-driven programming, or data oriented programming (DOP), is a specific programming paradigm where the data, and/or data structures, control the flow of a program and not the program logic. Often, data-driven programming approaches are applied in organizations with structured and unstructured data for filtering, aggregating, transforming and calling other programs. This paper and presentation explores several data-driven programming techniques that are available to SAS® users. Topics include using metadata to capture valuable information about a SAS session such as the librefs that are currently assigned, the names of the tables available in a session, whether a data set is empty, the number of observations in a data set, the number of character versus numeric variables in a data set, and a variable's attributes; using the CALL EXECUTE routine to process (or execute) code generated by a DATA step; constructing a user-defined format directly from data; and using the SQL procedure and the macro language to construct an automated looping process.

Introduction

The SAS System collects and populates valuable information ("metadata") about SAS libraries, data sets (tables), catalogs, indexes, macros, system options, titles, views and a collection of other read-only tables called dictionary tables. Dictionary tables serve a special purpose by providing system-related information about the current SAS session's SAS databases and applications. When a query is requested against a Dictionary table, SAS automatically launches a discovery process at runtime to collect information pertinent to that table. Metadata content can be very useful for developing data-driven techniques. Other data-driven programming techniques include using the CALL EXECUTE routine to process (or execute) code generated by a DATA step; constructing a user-defined format directly from data; and using the SQL procedure and the macro language to construct an automated looping process.

Tables Used in Examples

The data used in all the examples in this paper uses a movies and actors data set (table). The Movies table consists of twenty-two observations (rows) and six variables (columns): Title, Length, Category, Year, Studio, and Rating. Title, Category, Studio, and Rating are defined as character columns with Length and Year being defined as numeric columns. The Movies data set (table) is 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

The ACTORS data set (table) consists of thirteen observations (rows) and three variables (columns): Title, Actor_Leading, and Actor_Supporting which are all character columns, and is 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

Programming Paradigms

Programming languages are often classified by their basic features into one of the many programming paradigms. Three popular programming paradigms in use today by programming professionals are **1) Procedural programming** – represented by blocks of code being organized logically by function, such as data input, data processing or manipulation, and data / results output; **2) Object-oriented programming** – represented by a combination of functionality (behaviors) and data (attributes) hidden inside an object which can then be arranged into classes; and **3) Data-driven programming** – represented by data controlling the flow of execution in a program.

What is Data-driven Programming?

Unlike procedural programming languages where a program's flow of execution is described using a detailed step-by-step logical approach to solving a problem or with object-oriented programming where an object is told how to behave without all the detailed steps that informs the object how to behave. Data-driven programming involves a program that has its decisions and processes (the flow of execution) controlled (or dictated) by the data (or data structures).

Data-driven programming possesses many virtues over rival programming paradigms including having a default action assigned to it, provide greater flexibility, are often shorter in length, and can be easier to maintain due to a reduction, or elimination, of "hard-coded" values.

Traditional (or "Legacy") SAS Metadata Sources

SAS users have traditionally been accessing and producing metadata using PROC CONTENTS and PROC DATASETS.

- **PROC CONTENTS** – Produces a directory of the SAS library and the details associated with each member type stored in a SAS library.

- **PROC DATASETS** – In Michael A. Raithel's (2016) landmark paper, PROC DATASETS is the Swiss Army Knife of Data Management procedures. Like PROC CONTENTS, the PROC DATASETS CONTENTS statement produces a directory of the SAS library and the details associated with each member type (e.g., DATA, VIEW, INDEX) stored in a SAS library.

In the next example, PROC CONTENTS is specified to describe the metadata associated with the SAS data set, Movies.

PROC CONTENTS Code:

```
PROC CONTENTS DATA=WORK.Movies ;
RUN ;
```

Results from PROC CONTENTS:

The CONTENTS Procedure

Data Set Name	WORK.MOVIES	Observations	22
Member Type	DATA	Variables	6
Engine	V9	Indexes	0
Created	04/15/2018 04:58:10	Observation Length	88
Last Modified	04/15/2018 04:58:10	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	743
Obs in First Data Page	22
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	/tmp/SAS_work7E2E00006D27_localhost.localdomain/SAS_work32C300006D27_localhost.localdomain/movies.sas7bdat
Release Created	9.0401M5
Host Created	Linux
Inode Number	670869
Access Permission	rw-rw-r--
Owner Name	sasdemo
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	Category	Char	20
2	Length	Num	3
6	Rating	Char	5
5	Studio	Char	25
1	Title	Char	30
4	Year	Num	4

Sort Information	
Sortedby	Title
Validated	YES
Character Set	ANSI
Sort Option	NODUPKEY

In the next example, PROC CONTENTS is specified to print a list of all SAS files that reside in the SAS library.

PROC CONTENTS Code:

```
PROC CONTENTS DATA=WORK.Movies DIRECTORY ;
RUN ;
```

Results from PROC CONTENTS:

The CONTENTS Procedure

Directory	
Libref	WORK
Engine	V9
Physical Name	/tmp/SAS_work7E2E00006D27_localhost.localdomain/SAS_work32C300006D27_localhost.localdomain
Filename	/tmp/SAS_work7E2E00006D27_localhost.localdomain/SAS_work32C300006D27_localhost.localdomain
Inode Number	670832
Access Permission	rwx------
Owner Name	sasdemo
File Size	4KB
File Size (bytes)	4096

#	Name	Member Type	File Size	Last Modified
1	ACTORS	DATA	16KB	04/15/2018 11:58:10
2	MOVIES	DATA	128KB	04/15/2018 11:58:09
3	REGSTRY	ITEMSTOR	32KB	04/15/2018 11:52:49
4	SASGOPT	CATALOG	12KB	04/15/2018 11:58:09
5	SASMAC1	CATALOG	208KB	04/15/2018 11:52:49
6	SASMAC2	CATALOG	20KB	04/15/2018 11:52:49
7	SASMAC3	CATALOG	20KB	04/15/2018 11:52:49
8	SASMAC4	CATALOG	20KB	04/15/2018 12:05:27
9	SASMAC5	CATALOG	20KB	04/15/2018 11:52:49
10	SASMAC6	CATALOG	20KB	04/15/2018 11:52:49
11	SASMAC7	CATALOG	20KB	04/15/2018 11:52:49
12	SASMAC8	CATALOG	20KB	04/15/2018 11:52:49
13	SASMAC9	CATALOG	20KB	04/15/2018 11:52:49
14	SASMACR	CATALOG	20KB	04/15/2018 11:58:10

The CONTENTS Procedure

Data Set Name	WORK.MOVIES	Observations	22
Member Type	DATA	Variables	6
Engine	V9	Indexes	0
Created	04/15/2018 04:58:10	Observation Length	88
Last Modified	04/15/2018 04:58:10	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	743
Obs in First Data Page	22
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	/tmp/SAS_work7E2E00006D27_localhost.localdomain/SAS_work32C300006D27_localhost.localdomain/movies.sas7bdat
Release Created	9.0401M5
Host Created	Linux
Inode Number	670869
Access Permission	rw-rw-r--
Owner Name	sasdemo
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
3	Category	Char	20
2	Length	Num	3
6	Rating	Char	5
5	Studio	Char	25
1	Title	Char	30
4	Year	Num	4

Sort Information	
Sortedby	Title
Validated	YES
Character Set	ANSI
Sort Option	NODUPKEY

In the next example, PROC CONTENTS is specified to save the results of a SAS data set's metadata that resides in the SAS library to a SAS data set.

PROC CONTENTS and PROC PRINT Code:

```
PROC CONTENTS DATA=WORK.Movies
               OUT=WORK.Contents_Structure
              DIRECTORY ;
RUN ;
PROC PRINT DATA=WORK.Contents_Structure ;
RUN ;
```

Results from PROC CONTENTS and PROC PRINT:

< Same as PROC CONTENTS Results from the previous example >

. . . . . .

Obs	LIBNAME	MEMNAME	MEMLABEL	TYPEMEM	NAME	TYPE	LENGTH	VARNUM	LABEL	FORMAT	FORMATL	FORMATD	INFORMAT	INFORML	INFORMD
1	WORK	MOVIES			Category	2	20	3			0	0		0	0
2	WORK	MOVIES			Length	1	3	2			0	0		0	0
3	WORK	MOVIES			Rating	2	5	6			0	0		0	0
4	WORK	MOVIES			Studio	2	25	5			0	0		0	0
5	WORK	MOVIES			Title	2	30	1			0	0		0	0
6	WORK	MOVIES			Year	1	4	4			0	0		0	0

JUST	NPOS	NOBS	ENGINE	CRDATE	MODATE	DELOBS	IDXUSAGE	MEMTYPE	IDXCOUNT	PROTECT	FLAGS	COMPRESS	REUSE
0	37	22	V9	15APR18:04:58:10	15APR18:04:58:10	0	NONE	DATA	0	---	---	NO	NO
1	4	22	V9	15APR18:04:58:10	15APR18:04:58:10	0	NONE	DATA	0	---	---	NO	NO
0	82	22	V9	15APR18:04:58:10	15APR18:04:58:10	0	NONE	DATA	0	---	---	NO	NO
0	57	22	V9	15APR18:04:58:10	15APR18:04:58:10	0	NONE	DATA	0	---	---	NO	NO
0	7	22	V9	15APR18:04:58:10	15APR18:04:58:10	0	NONE	DATA	0	---	---	NO	NO
1	0	22	V9	15APR18:04:58:10	15APR18:04:58:10	0	NONE	DATA	0	---	---	NO	NO

SORTED	SORTEDBY	CHARSET	COLLATE	NODUPKEY	NODUPREC	ENCRYPT	POINTOBS	GENMAX	GENNUM	GENNEXT	TRANSCOD
1	.	ANSI		YES	NO	NO	YES	0	.	.	YES
1	.	ANSI		YES	NO	NO	YES	0	.	.	YES
1	.	ANSI		YES	NO	NO	YES	0	.	.	YES
1	.	ANSI		YES	NO	NO	YES	0	.	.	YES
1	1	ANSI		YES	NO	NO	YES	0	.	.	YES
1	.	ANSI		YES	NO	NO	YES	0	.	.	YES

Data-driven Programming Using DICTIONARY Tables and SASHELP Views

SAS users can quickly and conveniently obtain useful information (metadata) about their SAS session with a number of read-only SAS system tables called DICTIONARY tables and SASHELP views. At any time during a SAS session, DICTIONARY tables can be accessed using the libref DICTIONARY in the FROM clause of a PROC SQL SELECT statement to capture information related to currently defined libnames, table names, column names and attributes, formats, and much more. SASHELP views can be accessed using any of your favorite procedures or in the DATA step.

Identifying the Names of the DICTIONARIES Tables and SASHELP Views

SAS users can identify any new Dictionary table release by accessing the read-only DICTIONARIES Dictionary table or VSVIEW SASHELP view. The content of the DICTIONARIES Dictionary table reveals the names of supported Dictionary tables. The following PROC SQL query uses the UNIQUE (or DISTINCT) keyword to generate a listing of existing Dictionary tables.

PROC SQL Code:

```
PROC SQL ;
  SELECT UNIQUE MEMNAME
    FROM DICTIONARY.DICTIONARIES ;
QUIT ;
```

Results from DICTIONARY.DICTIONARIES:

Member Name
CATALOGS
CHECK_CONSTRAINTS
COLUMNS
CONSTRAINT_COLUMN_USAGE
CONSTRAINT_TABLE_USAGE
DATAITEMS
DESTINATIONS
DICTIONARIES
ENGINES
EXTFILES
FILTERS
FORMATS
FUNCTIONS
GOPTIONS
INDEXES
INFOMAPS

Member Name
LIBNAMES
LOCALES
MACROS
MEMBERS
OPTIONS
PROMPTS
PROMPTSXML
REFERENTIAL_CONSTRAINTS
REMEMBER
STYLES
TABLES
TABLE_CONSTRAINTS
TITLES
VIEWS
VIEW_SOURCES
XATTRS

SAS 9.4 currently supports 32 DICTIONARY tables as is illustrated below. Earlier versions of SAS supported fewer Dictionary tables. SAS 9.3 supported 30 DICTIONARY tables; SAS 9.2 supported 29 Dictionary tables; and SAS 9.1 software supported 22 Dictionary tables.

The contents of the VSVIEW SASHELP view reveals the names of supported SASHELP views in SAS 9.4. The following PROC SQL query uses the DISTINCT (or UNIQUE) keyword along with the SUBSTR function to identify a listing of SASHELP views starting with the character value, “V”.

PROC SQL Code:

```
PROC SQL ;
 SELECT DISTINCT MEMNAME
  FROM SASHELP.VSVIEW
   WHERE UPCASE(SUBSTR(MEMNAME,1,1)) = 'V'
     AND UPCASE(LIBNAME) = 'SASHELP'
    ORDER BY MEMNAME ;
QUIT ;
```

Results from SASHELP.VSVIEWS:

Member Name
VALLOPT
VCATALG
VCFORMAT
VCHKCON
VCNCOLU
VCNTABU
VCOLUMN
VDATAIT
VDCTNRY
VDEST
VENGINE
VEXTFL
VFILTER
VFORMAT
VFUNC
VGOPT
VINDEX
VINFOMP
VLIBNAM
VLOCALE
VMACRO
VMEMBER

Member Name
VOPTION
VPRMXML
VPROMPT
VREFCON
VREMEMB
VSACCES
VSCATLG
VSLIB
VSTABLE
VSTABVW
VSTYLE
VSVIEW
VTABCON
VTABLE
VTITLE
VVIEW
VXATTR

Names and Purpose of Each DICTIONARY Table and SASHELP View

The names and purpose of the DICTIONARY tables and equivalent SASHELP views appear in the following table.

DICTIONARY Table	SASHELP View	Purpose
CATALOGS	**VCATALG**	SAS Catalogs and Catalog-specific Information.
CHECK_CONSTRAINTS	**VCHKCON**	Check Constraints information.
COLUMNS	**VCOLUMN**	Columns from All Tables.
CONSTRAINT_COLUMN_USAGE	**VCNCOLU**	Constraint Column Usage.
CONSTRAINT_TABLE_USAGE	**VCNTABU**	Constraint Table Usage.
DATAITEMS	**VDATAIT**	Information Map Data Items.
DESTINATIONS	**VDEST**	Open ODS Destinations.
DICTIONARIES	**VDCTNRY**	DICTIONARY Tables and their Columns.
ENGINES	**VENGINE**	Available Engines.
EXTFILES	**VEXTFL**	Implicitly-defined File Definitions and Files Defined in FILENAME statements.
FILTERS	**VFILTER**	Information Map Filters.

FORMATS	**VFORMAT**	Available SAS and User-defined Formats and Informats.
FUNCTIONS	**VFUNC**	Available Functions.
GOPTIONS	**VGOPT**	SAS/GRAPH Software Graphics Options.
INDEXES	**VINDEX**	Information related to Defined Indexes.
INFOMAPS	**VINFOMP**	Information Maps.
LIBNAMES	**VLIBNAM**	Information related to SAS Data Libraries.
LOCALES	**VLOCALE**	Available Locales, Regions, Languages and Currency Symbols.
MACROS	**VMACRO**	Information about Defined Macros.
MEMBERS	**VMEMBER**	Information about SAS Defined Tables, Catalogs and Views.
OPTIONS	**VOPTION**	Information about SAS Default System Options.
PROMPTS	**VPROMPT**	Information about Information Map Prompts.
PROMPTSXML	**VPRMXML**	Information Map Prompts XML.
REFERENTIAL_CONSTRAINTS	**VREFCON**	Information about Referential Constraints.
REMEMBER	**VREMEMB**	All Remembered Information.
STYLES	**VSTYLE**	Information about All Styles.
TABLES	**VTABLE**	SAS Tables and Table-specific Information.
TABLE_CONSTRAINTS	**VTABCON**	Information about Table Constraints.
TITLES	**VTITLE**	Information about Defined Titles.
VIEWS	**VVIEW**	Views and View-specific Information.
VIEW_SOURCES	**VSVIEW**	Sources Referenced by View.
XATTRS	**VXATTR**	Extended Attributes.

Displaying DICTIONARY Table Definitions

A dictionary table's definition can be displayed by specifying a DESCRIBE TABLE statement. The results of the statements and clauses used to create each dictionary table can be displayed on the SAS Log. For example, a DESCRIBE TABLE statement is illustrated below to display the CREATE TABLE statement used in building the OPTIONS dictionary table containing current SAS System option settings.

PROC SQL Code:

```
PROC SQL ;
  DESCRIBE TABLE
    DICTIONARY.OPTIONS ;
QUIT ;
```

SAS Log Results:

```
create table DICTIONARY.OPTIONS
  (
   optname char(32) label='Option Name',
   setting char(1024) label='Option Setting',
   optdesc char(160) label='Option Description',
   level char(8) label='Option Location'
  );
```

Note: The information contained in dictionary tables is also available to DATA and PROC steps outside the SQL procedure. Referred to as SASHELP views, each view is prefaced with the letter "V" and may be shortened with abbreviated names. SASHELP views can be accessed by referencing the view by its name in the SASHELP library. Please refer to the SAS Procedures Guide for further details on accessing and using dictionary views in the SASHELP library.

The COLUMNS DICTIONARY Table and VCOLUMN SASHELP View

Retrieving information about the columns in one or more data sets or tables is easy with the COLUMNS dictionary table. Similar to the results of the CONTENTS procedure, users are able to capture column-level information including column name, type, length, position, label, format, informat, and indexes, as well as produce cross-reference listings containing the location of columns in a SAS library. For example, the following code requests a cross-reference listing of the tables containing the TITLE column in the WORK library. **Note:** Care should be used when specifying multiple functions on the WHERE clause since the SQL Optimizer is unable to optimize the query resulting in all allocated SAS session librefs being searched. This can cause the query to run much longer than expected.

PROC SQL Code:

```
PROC SQL ;
  SELECT *
    FROM DICTIONARY.COLUMNS
      WHERE UPCASE(LIBNAME)="WORK" AND
            UPCASE(NAME)="TITLE" ;
QUIT ;
```

Results:

Library Name	Member Name	Member Type	Column Name	Column Type	Column Length	Column Position	Column Number in Table	Column Label	Column Format	Column Informat	Column Index Type
Order in Key Sequence	Extended Type	Not NULL?	Precision	Scale	Transcoded?						
WORK	ACTORS	DATA	Title	char	30	0	1				
0	char	no	.	.	yes						
WORK	MOVIES	DATA	Title	char	30	7	1				SIMPLE
0	char	no	.	.	yes						

The TABLES DICTIONARY Table and VTABLE SASHELP View

When users need more information about SAS files consider using the TABLES Dictionary table or the VTABLE SASHELP view. The TABLES dictionary table provides detailed information about the library name, member name and type, date created and last modified, number of observations, observation length, number of variables, password protection, compression, encryption, number of pages, reuse space, buffer size, number of deleted observations, type of indexes, and requirements vector. For example, to obtain a detailed list of files in the WORK library, a PROC SQL SELECT query can be constructed as follows.

Note: Because the TABLE Dictionary table produces a considerable amount of information, users should consider specifying a WHERE clause when accessing this table.

PROC SQL Code:

```
PROC SQL ;
  SELECT *
    FROM DICTIONARY.TABLES
      WHERE UPCASE(LIBNAME)="WORK" ;
QUIT ;
```

Results:

Library Name	Member Name	Member Type	DBMS Member Type	Dataset Label	Dataset Type	Date Created	Date Modified	Number of Physical Observations
WORK	ACTORS	DATA			DATA	09AUG04:15:40:18	09AUG04:15:40:18	13
WORK	MOVIES	DATA			DATA	09AUG04:15:40:18	09AUG04:15:40:18	22

Observation Length	Number of Variables	Type of Password Protection	Compression Routine	Encryption	Number of Pages	Size of File	Percent Compression	Reuse Space	Bufsize
70	3	---	NO	NO	1	16384	0	no	8192
88	6	---	NO	NO	2	24576	0	no	8192

Number of Deleted Observations	Number of Logical Observations	Longest variable name	Longest label	Maximum number of generations	Generation number	Dataset Attributes	Type of Indexes	Data Representation
0	13	16	0	0	.	ON		NATIVE
0	22	8	0	0	.	ON	SIMPLE	NATIVE

Name of Collating Sequence	Sorting Type	Charset Sorted By	Requirements Vector	Data Representation Name	Data Encoding	Audit Trail Active?
			181F10112222003222010232043201222003E00001003 01	WINDOWS_32	wlatin1 Western (Windows)	no
			181F10112222003222010232043201222003E00001003 01	WINDOWS_32	wlatin1 Western (Windows)	no

Audit Before Image?	Audit Admin Image?	Audit Error Image?	Audit Data Image?
no	no	no	no
no	no	no	no

Accessing Information from SAS DICTIONARY Tables to Do Cool Things

SAS users can quickly and conveniently obtain useful information about their SAS session with a number of read-only SAS system tables called DICTIONARY tables. At any time during a SAS session, DICTIONARY tables can be accessed using the libref DICTIONARY in the FROM clause of a PROC SQL SELECT statement to capture information related to currently defined libnames, table names, column names and attributes, formats, and much more. SASHELP views can be accessed using any of your favorite procedures or in the DATA step. SAS 9.1 software supported 22 Dictionary tables and SASHELP views, SAS 9.2 supported 29 Dictionary tables and SASHELP views, SAS 9.3 supported 30 DICTIONARY tables and SASHELP views, and SAS 9.4 supports 32 DICTIONARY tables and SASHELP views.

Accessing and Displaying the Number of Rows in a Table

The DICTIONARY table, TABLES, can be accessed to capture and display each table name and the number of observations in the user-assigned WORK libref. The following PROC SQL code provides a handy way to quickly determine the number of rows in one or all tables in a libref without having to execute multiple PROC CONTENTS by using the stored information in the Dictionary table TABLES.

PROC SQL Code:

```
PROC SQL ;
  SELECT LIBNAME, MEMNAME, NOBS
    FROM DICTIONARY.TABLES
      WHERE UPCASE(LIBNAME)="WORK" AND
            UPCASE(MEMTYPE)="DATA" ;
QUIT ;
```

Results:

Library Name	Member Name	Number of Physical Observations
WORK	ACTORS	13
WORK	CUSTOMERS	3
WORK	MOVIES	22
WORK	PG_RATED_MOVIES	13

Accessing and Displaying the Column Definitions for a "Key" Variable (or Variables) in All Tables

The DICTIONARY table, COLUMNS, is accessed to display all table names (data sets) that contain the variable TITLE in the user-assigned WORK libref as a cross-reference listing. To retrieve the needed type of information, you could execute multiple PROC CONTENTS against selected tables. Or in a more efficient method, you could retrieve the information directly from the read-only Dictionary table COLUMNS with the selected columns LIBNAME, MEMNAME, NAME, TYPE and LENGTH, as shown. For more information about Dictionary tables, readers may want to view the "free" SAS Press Webinar by Kirk Paul Lafler at http://support.sas.com/publishing/bbu/webinar.html#lafler2 or the published paper by Kirk Paul Lafler, Exploring Dictionary Tables and SASHELP Views.

PROC SQL Code:

```
PROC SQL ;
  SELECT LIBNAME, MEMNAME, NAME, TYPE, LENGTH
    FROM DICTIONARY.COLUMNS
      WHERE UPCASE(LIBNAME)="WORK"
        AND UPCASE(NAME)="TITLE"
        AND UPCASE(MEMTYPE)="DATA" ;
QUIT ;
```

Results:

Library Name	Member Name	Column Name	Column Type	Column Length
WORK	ACTORS	Title	char	30
WORK	MOVIES	Title	char	30
WORK	PG_MOVIES	Title	char	30
WORK	PG_RATED_MOVIES	Title	char	30
WORK	RENTAL_INFO	Title	char	30

Capturing a List of Variables from the COLUMNS Dictionary Table

The DICTIONARY table, COLUMNS, can be accessed to capture and display each column name contained in one or more tables in the WORK libref. The following PROC SQL code provides a handy way to quickly capture the names of any, and all, columns contained in the MOVIES table without having to execute PROC CONTENTS.

PROC SQL Code:

```
PROC SQL NOPRINT ;
  SELECT NAME,
         COUNT(NAME)
         INTO :MVARIABLES SEPARATED BY ' ',
              :MVARIABLESNUM
   FROM DICTIONARY.COLUMNS
    WHERE UPCASE(LIBNAME)="WORK"
      AND UPCASE(MEMNAME)="MOVIES" ;
QUIT ;
%PUT &MVARIABLES &MVARIABLESNUM ;
```

SAS Log Results:

```
%PUT &MVARIABLES &MVARIABLESNUM ;
Title Length Category Year Studio Rating        6
```

The previous example can be expanded so only the character-defined variables are saved in the macro variable. The next example illustrates PROC SQL code to capture the names of the character-defined columns contained in the MOVIES table and the contents of the macro variable is then specified in a SELECT statement to produce a report.

PROC SQL Code:

```
PROC SQL NOPRINT ;
  SELECT NAME
         INTO :MVARIABLES SEPARATED BY ', '
   FROM DICTIONARY.COLUMNS
    WHERE UPCASE(LIBNAME)="WORK"
      AND UPCASE(MEMNAME)="MOVIES"
      AND UPCASE(TYPE)="CHAR" ;
  %PUT &MVARIABLES ;
  RESET PRINT ;
  SELECT &MVARIABLES FROM MOVIES ;
QUIT ;
```

SAS Log Results:

```
%PUT &MVARIABLES ;
Title, Category, Studio, Rating
```

PROC PRINT Results:

Title	Category	Studio	Rating
Brave Heart	Action Adventure	Paramount Pictures	R
Casablanca	Drama	MGM / UA	PG
Christmas Vacation	Comedy	Warner Brothers	PG-13
Coming to America	Comedy	Paramount Pictures	R
Dracula	Horror	Columbia TriStar	R
Dressed to Kill	Drama Mysteries	Filmways Pictures	R
Forrest Gump	Drama	Paramount Pictures	PG-13
Ghost	Drama Romance	Paramount Pictures	PG-13
Jaws	Action Adventure	Universal Studios	PG
Jurassic Park	Action	Universal Pictures	PG-13
Lethal Weapon	Action Cops & Robber	Warner Brothers	R
Michael	Drama	Warner Brothers	PG-13
National Lampoon's Vacation	Comedy	Warner Brothers	PG-13
Poltergeist	Horror	MGM / UA	PG
Rocky	Action Adventure	MGM / UA	PG
Scarface	Action Cops & Robber	Universal Studios	R
Silence of the Lambs	Drama Suspense	Orion	R
Star Wars	Action Sci-Fi	Lucas Film Ltd	PG
The Hunt for Red October	Action Adventure	Paramount Pictures	PG
The Terminator	Action Sci-Fi	Live Entertainment	R
The Wizard of Oz	Adventure	MGM / UA	G
Titanic	Drama Romance	Paramount Pictures	PG-13

Data-driven Programming Using the CALL EXECUTE Routine

SAS users have a powerful DATA step routine called CALL EXECUTE that can be used for data-driven processing. The CALL EXECUTE routine accepts a single argument where the value can be a character-string or, when needed, a character expression containing SAS code elements to be executed after they are resolved. The CALL EXECUTE routine permits SAS statements and macro code to be stacked together and then executed, Batkhan (2017).

When the CALL EXECUTE routine contains SAS statement code without macro variables or macro references, the code is appended to the input stack for immediate execution after the DATA step ends. The argument can be specified with single or double quotes, dynamically generating SAS code for execution. To leverage data-driven processes with CALL EXECUTE, a control data set containing four distinct movie ratings (i.e., “G”, “PG”, “PG-13” and “R”) represented as four observations is created. The second DATA step then reads the contents of the control data set populating the unique value for the movie_rating variable in the individual CALL EXECUTE statements. **Note:** The CATS function is used to strip blanks and concatenate multiple strings together.

Code:

```
data movie_list ;  /* Control Data Set */
  length movie_rating $5. ;
  input @1 movie_rating $5. ;
  datalines ;
G
PG
PG-13
R
;
run ;

data _null_ ;  /* Populate Unique Value for the Movie_rating Variable */
```

```
  set movie_list ;
  call execute(CATS('ods Excel file="/folders/myfolders/',movie_rating,'_Rpt.xlsx"
                            style=styles.barrettsblue
                            options(embedded_titles="yes");')) ;
  call execute(CAT('title ', movie_rating, ' Movies;')) ;
  call execute(CATS('proc freq data=mydata.Movies(where=
                           (rating="',movie_rating,'"));')) ;
  call execute('tables Title;') ;
  call execute('run;') ;
  call execute('ods Excel close;') ;
run ;
```

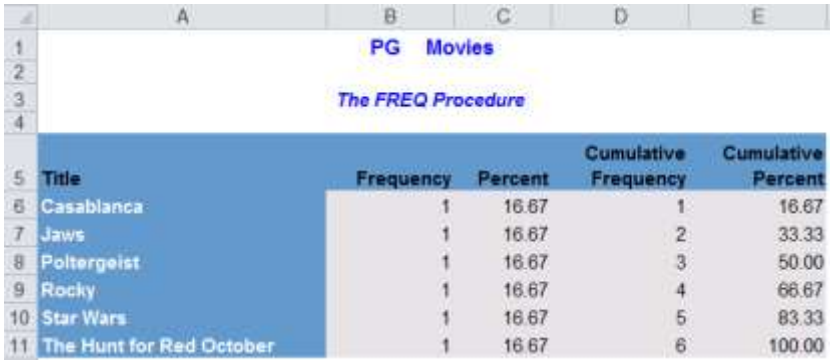
The dynamically generated SAS code produced by the preceding CALL EXECUTE statements is displayed, below.

SAS Log (Generated SAS Code):

```
1  + ods Excel file="/folders/myfolders/G_Rpt.xlsx"
             style=styles.barrettsblue
             options(embedded_titles="yes");
2  + title G  Movies;
3  + proc freq data=mydata.Movies(where=(rating="G"));
4  + tables Title;
5  + run;

6  + ods Excel file="/folders/myfolders/PG_Rpt.xlsx"
             style=styles.barrettsblue
             options(embedded_titles="yes");
7  + title PG  Movies;
8  + proc freq data=mydata.Movies(where=(rating="PG"));
9  + tables Title;
10 + run;

11 + ods Excel file="/folders/myfolders/PG-13_Rpt.xlsx"
             style=styles.barrettsblue
             options(embedded_titles="yes");
12 + title PG-13  Movies;
13 + proc freq data=mydata.Movies(where=(rating="PG-13"));
14 + tables Title;
15 + run;

16 + ods Excel file="/folders/myfolders/R_Rpt.xlsx"
             style=styles.barrettsblue
             options(embedded_titles="yes");
17 + title R  Movies;
18 + proc freq data=mydata.Movies(where=(rating="R"));
19 + tables Title;
20 + run;
```

Results (Using CALL EXECUTE):

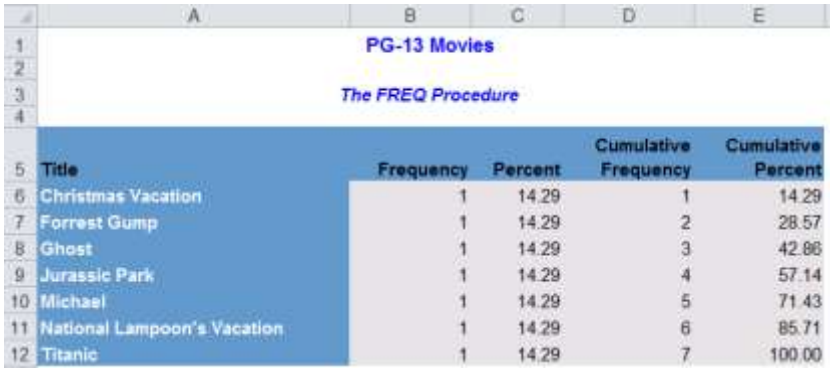
	A	B	C	D	E
1			G Movies		
2					
3			*The FREQ Procedure*		
4					
5	**Title**	**Frequency**	**Percent**	**Cumulative Frequency**	**Cumulative Percent**
6	The Wizard of Oz	1	100.00	1	100.00



PG Movies

The FREQ Procedure

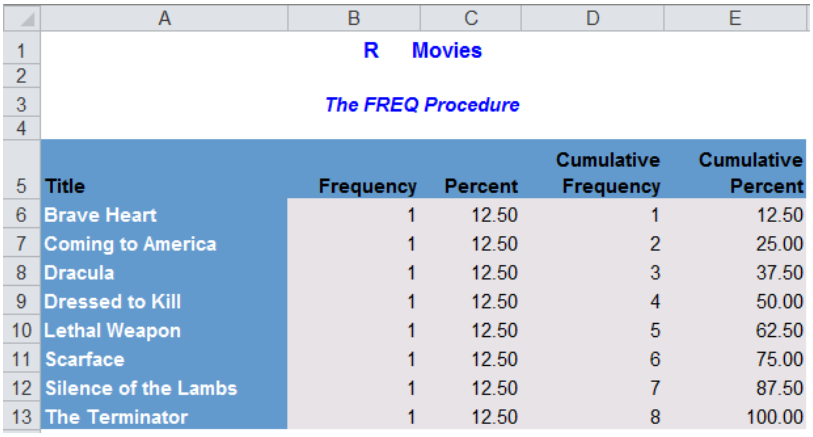
Title	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Casablanca	1	16.67	1	16.67
Jaws	1	16.67	2	33.33
Poltergeist	1	16.67	3	50.00
Rocky	1	16.67	4	66.67
Star Wars	1	16.67	5	83.33
The Hunt for Red October	1	16.67	6	100.00



PG-13 Movies

The FREQ Procedure

Title	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Christmas Vacation	1	14.29	1	14.29
Forrest Gump	1	14.29	2	28.57
Ghost	1	14.29	3	42.86
Jurassic Park	1	14.29	4	57.14
Michael	1	14.29	5	71.43
National Lampoon's Vacation	1	14.29	6	85.71
Titanic	1	14.29	7	100.00



R Movies

The FREQ Procedure

Title	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Brave Heart	1	12.50	1	12.50
Coming to America	1	12.50	2	25.00
Dracula	1	12.50	3	37.50
Dressed to Kill	1	12.50	4	50.00
Lethal Weapon	1	12.50	5	62.50
Scarface	1	12.50	6	75.00
Silence of the Lambs	1	12.50	7	87.50
The Terminator	1	12.50	8	100.00

Data-driven Programming Using Custom-defined Formats from Data

The FORMAT procedure is a powerful tool for building user-defined informats and formats. These "custom" user-defined informats and formats provides SAS with instructions on how to read data into SAS variables and write (or display) output. Custom-defined informats and formats are defined as temporary or permanent, and are stored as entries in SAS catalogs. The available operations performed by the FORMAT procedure include the ability to:

- Convert character values to numeric values
- Convert numeric values to character values
- Convert character values to other character values.

To prevent hard-coding VALUE clauses, custom-defined formats can be created dynamically from a SAS data set. This not only can be a more efficient approach than processing sort, merge, and join operations, Droogendyk (2010), it also allows data-driven processes to be leveraged. Consequently, the FORMAT procedure is able to create informats and formats without specifying an INVALUE, PICTURE, or VALUE clause by using a SAS control data set as input.

The control data set is specified with the **CNTLIN** option of PROC FORMAT. To start the process, the control data set being used must have the following required variables:

- FMTNAME - specifies the name of a character variable whose value is the format or informat name.
- START - specifies the name of a character variable that contains the value to be converted.
- LABEL - specifies the name of a character variable that contains the converted value.

In the next example, the DATA step is specified with IF-THEN/ELSE logic to produce the control data set with the required variables. The contents of the control data set are then displayed with the PRINT procedure. Finally, the control data set is specified in the PROC FORMAT CNTLIN option.

Code:

```
data control ;
  fmtname = "$Movie_Rating" ;
  length label $23. ;
  input start $5. ;
  if start = "G"          then label = "General Audience" ;
  else if start = "PG"    then label = "Parental Guidance" ;
  else if start = "PG-13" then label = "Parental Guidance >= 13" ;
  else if start = "R"     then label = "Restricted" ;
  else if start = ""      then label = "ERROR - Movie Rating" ;
  output ;
  datalines ;
G
PG
PG-13
R

;
run ;

proc print data=control noobs ;
  title ;
  var fmtname start label ;
run ;

proc format library=work cntlin=control ;
quit ;

proc print data=mydata.movies noobs ;
  format rating $movie_rating. ;
run ;
```

SAS Log (Generated Control Data Set):

```
data control ;
  fmtname = "$Movie_Rating" ;
  length label $23. ;
  input start $5. ;
  if start = "G"          then label = "General Audience" ;
  else if start = "PG"    then label = "Parental Guidance" ;
  else if start = "PG-13" then label = "Parental Guidance >= 13" ;
  else if start = "R"     then label = "Restricted" ;
  else if start = ""      then label = "ERROR - Movie Rating" ;
  output ;
  datalines ;
```

```
 NOTE: The data set WORK.CONTROL has 5 observations and 3 variables.

;
run ;

proc print data=control noobs ;
  title ;
  var fmtname start label ;
run ;

 NOTE: There were 5 observations read from the data set WORK.CONTROL.

proc format library=work cntlin=control ;
 NOTE: Format $MOVIE_RATING has been output.
quit ;

 NOTE: There were 5 observations read from the data set WORK.CONTROL.

proc print data=mydata.movies noobs ;
  format rating $movie_rating. ;
run ;

 NOTE: There were 22 observations read from the data set MYDATA.MOVIES.
```

Results:

The results of printing the dynamically-generated control data set and performing the formatted output are displayed, below.

fmtname	start	label
$Movie_Rating	G	General Audience
$Movie_Rating	PG	Parental Guidance
$Movie_Rating	PG-13	Parental Guidance >= 13
$Movie_Rating	R	Restricted
$Movie_Rating		ERROR – Movie Rating

Title	Length	Category	Year	Studio	Rating
Brave Heart	177	Action Adventure	1995	Paramount Pictures	Restricted
Casablanca	103	Drama	1942	MGM / UA	Parental Guidance
Christmas Vacation	97	Comedy	1989	Warner Brothers	Parental Guidance >= 13
Coming to America	116	Comedy	1988	Paramount Pictures	Restricted
Dracula	130	Horror	1993	Columbia TriStar	Restricted
Dressed to Kill	105	Drama Mysteries	1980	Filmways Pictures	Restricted
Forrest Gump	142	Drama	1994	Paramount Pictures	Parental Guidance >= 13
Ghost	127	Drama Romance	1990	Paramount Pictures	Parental Guidance >= 13
Jaws	125	Action Adventure	1975	Universal Studios	Parental Guidance
Jurassic Park	127	Action	1993	Universal Pictures	Parental Guidance >= 13
Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	Restricted
Michael	106	Drama	1997	Warner Brothers	Parental Guidance >= 13
National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	Parental Guidance >= 13
Poltergeist	115	Horror	1982	MGM / UA	Parental Guidance
Rocky	120	Action Adventure	1976	MGM / UA	Parental Guidance
Scarface	170	Action Cops & Robber	1983	Universal Studios	Restricted
Silence of the Lambs	118	Drama Suspense	1991	Orion	Restricted
Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	Parental Guidance
The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	Parental Guidance
The Terminator	108	Action Sci-Fi	1984	Live Entertainment	Restricted
The Wizard of Oz	101	Adventure	1939	MGM / UA	General Audience
Titanic	194	Drama Romance	1997	Paramount Pictures	Parental Guidance >= 13

Data-driven Programming Using the SQL Procedure and the Macro Language

The SQL procedure and the macro language are two versatile tools found in the Base SAS software. Combining the two together provides users with all the tools necessary to construct highly useful and effective data-driven programs. Lafler (2018) offers a data-driven approach to creating multiple Excel files. Triggered by calling a macro to reduce coding requirements, the process uses the Macro language, PROC SQL, the ODS Excel destination, and PROC FREQ to send output (results) to Excel. The **ODS Excel** Destination became production in SAS 9.4 (M4). It serves as an interface between SAS and Excel. The ODS Excel features include:

- ✓ SAS Results and Output can be sent directly to Excel
- ✓ Offers a Flexible way to create Excel files
- ✓ Supports Reports, Tables, Statistics and Graphs
- ✓ Formats Data into Excel Worksheet cells
- ✓ Permits Automation of Production-level Workbooks.

The ODS Excel destination easily sends output and results to Excel. The ODS Excel syntax simplifies the process of sending output, reports, tables, statistics and graphs to Excel files. The ODS Excel options are able to:

- ✓ Programmatically generate output and results
- ✓ Control font used and font sizes
- ✓ Add special features to row and column headers
- ✓ Adjust row and column sizes
- ✓ Format data values
- ✓ Align data to the left, center or right
- ✓ Add hyperlinks for drill-down capability.

Producing Multiple Excel Files

In the next example, a data-driven approach using PROC SQL SELECT code embedded inside a user-defined macro routine is constructed to dynamically produce separate Excel spreadsheets containing the frequency results for each unique By-group (e.g., Movie Rating). The SELECT query processes the Movies table, creates a single-value macro variable with the number of unique movie ratings, and a value-list macro variable with a list of the unique movie ratings separated with a tilde "~". Using the FREQ procedure and a user-defined macro, both macro variables along with their respective values, an iterative macro %DO statement, a %SCAN function, and WHERE= data set option dynamically sends the results to one or more Excel spreadsheets for each By-group.

Macro and PROC SQL Code:

```
%macro multExcelfiles ;
  proc sql noprint ;
   select count(distinct rating)
    into :mrating_cnt  /* number of unique movie ratings */
     from WORK.Movies
      order by rating ;
   select distinct rating
    into :mrating_lst separated by "~"  /* list of movies */
     from WORK.Movies
      order by rating ;
  quit ;
  %do i=1 %to &mrating_cnt ;
     ods Excel file="c:/%SCAN(&mrating_lst,&i,~)_Rpt.xlsx"
               style=styles.barrettsblue
               options(embedded_titles="yes") ;
     title "%SCAN(&mrating_lst,&i,~)-Rated Movies" ;
     proc freq data=WORK.Movies(where=(rating="%SCAN(&mrating_lst,&i,~)")) ;
       tables Title ;
     run ;
```

```
      ods Excel close ;
    %end ;
    %put &mrating_lst ;
  %mend multExcelfiles ;

  %multExcelfiles ;
```

The dynamically generated SAS code produced by the iterative %DO statement is displayed, below.

SAS Log (Generated SAS Code):

```
ods Excel file="/folders/myfolders/G_Rpt.xlsx"
         style=styles.barrettsblue
         options(embedded_titles="yes") ;
title "G-Rated Movies" ;
proc freq data=mydata.Movies(where=(rating="G")) ;
  tables Title ;
run ;
ods Excel close ;
NOTE: There were 1 observations read from the data set MYDATA.MOVIES.
       WHERE rating='G';
NOTE: Writing EXCEL file: /folders/myfolders/G_Rpt.xlsx


ods Excel file="/folders/myfolders/PG_Rpt.xlsx"
         style=styles.barrettsblue
         options(embedded_titles="yes") ;
title "PG-Rated Movies" ;
proc freq data=mydata.Movies(where=(rating="PG")) ;
  tables Title ;
run ;
ods Excel close ;
NOTE: There were 6 observations read from the data set MYDATA.MOVIES.
       WHERE rating='PG';
NOTE: Writing EXCEL file: /folders/myfolders/PG_Rpt.xlsx


ods Excel file="/folders/myfolders/PG-13_Rpt.xlsx"
         style=styles.barrettsblue
         options(embedded_titles="yes") ;
title "PG-13-Rated Movies" ;
proc freq data=mydata.Movies(where=(rating="PG-13")) ;
  tables Title ;
run ;
ods Excel close ;
NOTE: There were 7 observations read from the data set MYDATA.MOVIES.
       WHERE rating='PG-13';
NOTE: Writing EXCEL file: /folders/myfolders/PG-13_Rpt.xlsx


ods Excel file="/folders/myfolders/R_Rpt.xlsx"
         style=styles.barrettsblue
         options(embedded_titles="yes") ;
title "R-Rated Movies" ;
proc freq data=mydata.Movies(where=(rating="R")) ;
  tables Title ;
run ;
ods Excel close ;
```

```
NOTE: There were 8 observations read from the data set MYDATA.MOVIES.
      WHERE rating='R';
NOTE: Writing EXCEL file: /folders/myfolders/R_Rpt.xlsx

G~PG~PG-13~R
```

Results (4 Excel spreadsheets are produced):

G-Rated Movies

The FREQ Procedure

Title	Frequency	Percent	Cumulative Frequency	Cumulative Percent
The Wizard of Oz	1	100.00	1	100.00

PG-Rated Movies

The FREQ Procedure

Title	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Casablanca	1	16.67	1	16.67
Jaws	1	16.67	2	33.33
Poltergeist	1	16.67	3	50.00
Rocky	1	16.67	4	66.67
Star Wars	1	16.67	5	83.33
The Hunt for Red October	1	16.67	6	100.00

PG-13-Rated Movies

The FREQ Procedure

Title	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Christmas Vacation	1	14.29	1	14.29
Forrest Gump	1	14.29	2	28.57
Ghost	1	14.29	3	42.86
Jurassic Park	1	14.29	4	57.14
Michael	1	14.29	5	71.43
National Lampoon's Vacation	1	14.29	6	85.71
Titanic	1	14.29	7	100.00

R-Rated Movies

The FREQ Procedure

Title	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Brave Heart	1	12.50	1	12.50
Coming to America	1	12.50	2	25.00
Dracula	1	12.50	3	37.50
Dressed to Kill	1	12.50	4	50.00
Lethal Weapon	1	12.50	5	62.50
Scarface	1	12.50	6	75.00
Silence of the Lambs	1	12.50	7	87.50
The Terminator	1	12.50	8	100.00

Conclusion

Unlike procedural programming languages where a program's flow of execution is described using a detailed step-by-step logical approach to solving a problem or with object-oriented programming where an object is told how to behave, data-driven programming involves writing code that has its decisions and processes (the flow of execution) controlled (or dictated) by the data (or data structures). Data-driven programming offers SAS users with many virtues over rival programming paradigms including enhanced flexibility and easier to maintain due to a reduction, or elimination, of "hard-coded" values. Several data-driven programming techniques were presented including using the SAS System's read-only Dictionary tables and SASHELP views to provide valuable information about SAS libraries, data sets, columns and attributes, catalogs, indexes, macros, system options, and views; using the CALL EXECUTE routine to process (or execute) code generated by a DATA step; constructing a user-defined format directly from data; and using the SQL procedure and the macro language to construct an automated looping process.

References

Abolafia, Jeff and Frank DiIorio (2008), *"*Building Intelligent Macros: Using Metadata Functions with the SAS® Macro Language*,"* Proceedings of the 2008 SAS Global Forum (SGF) Conference.

Batkhan, Leonid, 2017, "CALL EXECUTE made easy for SAS data-driven programming", a SAS Blog Post.

Batkhan, Leonid, 2016, "Modifying variable attributes in all datasets of a SAS library", a SAS Blog Post, http://blogs.sas.com/content/sgf/2016/11/25/modifying-variable-attributes-in-all-datasets-of-a-sas-library/.

Carpenter, Arthur L. (2017), *"*Building Intelligent Macros: Using Metadata Functions with the SAS® Macro Language*,"* 2017 SAS Global Forum (SGF) Conference, California Occidental Consultants, Anchorage, AK, USA.

Davis, Michael (2001), *"You Could Look It Up: An Introduction to SASHELP Dictionary Views,"* Proceedings of the 2001 SAS Users Group International (SUGI) Conference, Bassett Consulting Services, North Haven, CT, USA.

Droogendyk, Harry (2010). "SAS® Formats: Effective and Efficient," Proceedings of the 2010 SouthEast SAS Users Group (SESUG) Conference.

Graebner, Robert W. (2001). "Developing Data-Driven SAS® Programs Using PROC CONTENTS," Proceedings of the 2001 MidWest SAS Users Group (MWSUG) Conference.

Hamilton, Jack (1998), "*Some Utility Applications of the Dictionary Tables in PROC SQL*," Proceedings of the 1998 Western Users of SAS Software (WUSS) Conference, 85-90.

Lafler, Kirk Paul (2016), *"Valuable Things You Can Do with SAS DICTIONARY Tables and SASHELP Views,"* Wisconsin Illinois SAS Users (WIILSU) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

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

Lafler, Kirk Paul (2012), *"Exploring DICTIONARY Tables and SASHELP Views,"* South Central SAS Users Group (SCSUG) Conference and Kansas City SAS Users Group (KCSUG) Meeting, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2009), *"DATA Step versus PROC SQL Programming Techniques,"* 2009 South East SAS Users Group (SESUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2009), *"Exploring DICTIONARY Tables and SASHELP Views,"* 2009 Western Users of SAS Software (WUSS) Conference and 2009 Pharmaceutical SAS Users Group (PharmaSUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2008), *"Undocumented and Hard-to-find PROC SQL Features,"* Greater Atlanta SAS Users Group (GASUG) Meeting (June 11th, 2008); Pharmaceutical SAS Users Group (PharmaSUG) Conference (June 1st - 4th, 2008); 2008 Michigan SAS Users Group (MSUG) Meeting (May 29th, 2008); 2008 Vancouver SAS Users Group Meeting (April 23rd, 2008); and 2008 PhilaSUG User Group Meeting (March 13th, 2008); Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2006), "*Exploring Dictionary Tables with PROC SQL*," SAS Press Webinar Series – June 27, 2006.

Lafler, Kirk Paul (2005), "*Exploring Dictionary Tables and SASHELP Views*," Proceedings of the Thirteenth Annual Western Users of SAS Software Conference.

Matise, Joe (2016). "Writing Code With Your Data: Basics of Data-Driven Programming Techniques," Proceedings of the 2016 South East SAS Users Group (SESUG) Conference.

Raithel, Michael A. (2016). "PROC DATASETS; The Swiss Army Knife of SAS® Procedures," Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Varney, Brian (2000). "How to Think Through the SAS® DATA Step," Proceedings of the 2000 SAS Users Group International (SUGI) Conference.

Villacorte, Renato G. (2012). "Go Beyond The Wizard With Data-Driven Programming," Proceedings of the 2012 SAS Global Forum (SGF) Conference.

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

Whitlock, Ian (2006). "How to Think Through the SAS® DATA Step," Proceedings of the 2006 SAS Users Group International (SUGI) Conference.

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

Acknowledgments

The author thanks Kelly Woodall and Michael Johnston, Beyond the Basics Section Chairs, for accepting my abstract and paper; Nina Worden, WUSS 2018 Academic Program Chair; Andra Northup, WUSS 2018 Operations Chair; the Western Users of SAS Software (WUSS) Executive Board; and SAS Institute for organizing and supporting a great conference!

Trademark Citations

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

About The Author

Kirk Paul Lafler is an entrepreneur, consultant and founder at Software Intelligence Corporation, and has been using SAS since 1979. Kirk is a SAS application developer, programmer, certified professional, provider of SAS consulting and application development services, mentor, advisor and adjunct professor at University of California San Diego Extension, emeritus sasCommunity.org Advisory Board member, and educator to SAS users around the world. As the author of six books including Google® Search Complete (Odyssey Press. 2014) and PROC SQL: Beyond the Basics Using SAS, Second Edition (SAS Press. 2013); Kirk has written hundreds of papers and articles; served as an Invited speaker, trainer, keynote and section leader at SAS user group conferences and meetings worldwide; and is the recipient of 25 "Best" contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions can be sent to:

Kirk Paul Lafler
SAS® Consultant, Application Developer, Programmer, Data Analyst, Educator and Author
Software Intelligence Corporation
E-mail: KirkLafler@cs.com
LinkedIn: https://www.linkedin.com/in/KirkPaulLafler
Twitter: @sasNerd