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.
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.

<table>
<thead>
<tr>
<th>Title</th>
<th>Actor_Leading</th>
<th>Actor_Supporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brave Heart</td>
<td>Mel Gibson</td>
<td>Sophia Maccuio</td>
</tr>
<tr>
<td>Christmas Vacation</td>
<td>Chris Chace</td>
<td>Beverly D’Angelo</td>
</tr>
<tr>
<td>Coming to America</td>
<td>Eddie Murphy</td>
<td>Anwar Ehsan</td>
</tr>
<tr>
<td>Forrest Gump</td>
<td>Tom Hanks</td>
<td>Sally Field</td>
</tr>
<tr>
<td>Ghost</td>
<td>Patrick Swayze</td>
<td>Dana Moore</td>
</tr>
<tr>
<td>Lethal Weapon</td>
<td>Mel Gibson</td>
<td>Danny Glover</td>
</tr>
<tr>
<td>Michael</td>
<td>John Travolta</td>
<td>Amanda MacDowell</td>
</tr>
<tr>
<td>National Lampoon's Vacation</td>
<td>Chris Chace</td>
<td>Beverly D'Angelo</td>
</tr>
<tr>
<td>Rocky</td>
<td>Sylvester Stallone</td>
<td>Taia Shire</td>
</tr>
<tr>
<td>Silence of the Lambs</td>
<td>Anthony Hopkins</td>
<td>Jodie Foster</td>
</tr>
<tr>
<td>The Hunt for Red October</td>
<td>Sean Connery</td>
<td>Alec Baldwin</td>
</tr>
<tr>
<td>The Terminator</td>
<td>Arnold Schwarzenegger</td>
<td>Michael Biehn</td>
</tr>
<tr>
<td>Titanic</td>
<td>Leonardo DiCaprio</td>
<td>Kate Winslet</td>
</tr>
</tbody>
</table>

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:**

```sas
PROC CONTENTS DATA=WORK.Movies ;
RUN ;
```
Results from PROC CONTENTS:

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:
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:**

```sas
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:**

```plaintext
< Same as PROC CONTENTS Results from the previous example >
```

---

**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 V$VIEW 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:**

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

**Results from DICTIONARY.DICTIONARIES:**

<table>
<thead>
<tr>
<th>Member Name</th>
<th>MemName</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOGS</td>
<td>LIBNAME</td>
</tr>
<tr>
<td>CHECK_CONSTRAINTS</td>
<td>LOCALES</td>
</tr>
<tr>
<td>COLUMNS</td>
<td>MACROS</td>
</tr>
<tr>
<td>CONSTRAINT_COLUMN_USAGE</td>
<td>MEMBERS</td>
</tr>
<tr>
<td>CONSTRAINT_TABLE_USAGE</td>
<td>OPTIONS</td>
</tr>
<tr>
<td>DATASETS</td>
<td>PROMPTS</td>
</tr>
<tr>
<td>DESTINATIONS</td>
<td>PROMPTXML</td>
</tr>
<tr>
<td>DICTIONARIES</td>
<td>REFERENCE_CONSTRAINTS</td>
</tr>
<tr>
<td>ENGINES</td>
<td>REMEMBER</td>
</tr>
<tr>
<td>EXTFILES</td>
<td>STYLES</td>
</tr>
<tr>
<td>FILTERS</td>
<td>TABLES</td>
</tr>
<tr>
<td>FORMATS</td>
<td>TABLE_CONSTRAINTS</td>
</tr>
<tr>
<td>FUNCTIONS</td>
<td>TITLES</td>
</tr>
<tr>
<td>GOPTIONS</td>
<td>VIEWS</td>
</tr>
<tr>
<td>INDEXES</td>
<td>VIEW_SOURCES</td>
</tr>
<tr>
<td>INFORMAPS</td>
<td>XATRAS</td>
</tr>
</tbody>
</table>

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 V$VIEW 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:**

```sql
PROC SQL ;
  SELECT DISTINCT MEMNAME
  FROM SASHELP.V$VIEW
  WHERE UPCASE(SUBSTR(MEMNAME,1,1)) = 'V'
    AND UPCASE(LIBNAME) = 'SASHELP'
  ORDER BY MEMNAME ;
QUIT ;
```
Results from SASHELP.VSVIEWS:

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Member Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALLOPT</td>
<td>VOPTION</td>
</tr>
<tr>
<td>VCATALG</td>
<td>VFPMXML</td>
</tr>
<tr>
<td>VFORMAT</td>
<td>VFDELETE</td>
</tr>
<tr>
<td>VCHKCON</td>
<td>VREFCON</td>
</tr>
<tr>
<td>VCNCOLU</td>
<td>VREM(cond)</td>
</tr>
<tr>
<td>VCNTABU</td>
<td>VSACCES</td>
</tr>
<tr>
<td>VOCOLUMN</td>
<td>VSCATLG</td>
</tr>
<tr>
<td>VDATAIT</td>
<td>VSLIB</td>
</tr>
<tr>
<td>VDCTRNY</td>
<td>VSTABLE</td>
</tr>
<tr>
<td>VDEST</td>
<td>VSTABFW</td>
</tr>
<tr>
<td>VENGINE</td>
<td>VSTYLE</td>
</tr>
<tr>
<td>VEXTFL</td>
<td>VSVIEW</td>
</tr>
<tr>
<td>VFILTER</td>
<td>VTABCON</td>
</tr>
<tr>
<td>VFORMAT</td>
<td>VTABLE</td>
</tr>
<tr>
<td>VFUNC</td>
<td>VTITLE</td>
</tr>
<tr>
<td>VGOPT</td>
<td>VVIEW</td>
</tr>
<tr>
<td>VINDEX</td>
<td>VXATTR</td>
</tr>
</tbody>
</table>

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.

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

```sql
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 prefixed 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:

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Member Name</th>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Length</th>
<th>Column Position</th>
<th>Column Number in Table</th>
<th>Column Label</th>
<th>Column Format</th>
<th>Column Informat</th>
<th>Column Index Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK</td>
<td>ACTORS</td>
<td>DATA</td>
<td>Title</td>
<td>char</td>
<td>30</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>SIMPLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORK</td>
<td>MOVIES</td>
<td>DATA</td>
<td>Title</td>
<td>char</td>
<td>30</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>SIMPLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

![Table Screenshot]

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.
Introduction to Data-driven Programming Techniques Using SAS®, continued

PROC SQL Code:

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

Results:

<table>
<thead>
<tr>
<th>Library</th>
<th>Member Name</th>
<th>Number of Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK</td>
<td>ACTORS</td>
<td>13</td>
</tr>
<tr>
<td>WORK</td>
<td>CUSTOMERS</td>
<td>3</td>
</tr>
<tr>
<td>WORK</td>
<td>MOVIES</td>
<td>22</td>
</tr>
<tr>
<td>WORK</td>
<td>PG_RATED_MOVIES</td>
<td>13</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Library</th>
<th>Member Name</th>
<th>Column Name</th>
<th>Column Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK</td>
<td>ACTORS</td>
<td>Title</td>
<td>char</td>
<td>30</td>
</tr>
<tr>
<td>WORK</td>
<td>MOVIES</td>
<td>Title</td>
<td>char</td>
<td>30</td>
</tr>
<tr>
<td>WORK</td>
<td>PG_MOVIES</td>
<td>Title</td>
<td>char</td>
<td>30</td>
</tr>
<tr>
<td>WORK</td>
<td>PG_RATED_MOVIES</td>
<td>Title</td>
<td>char</td>
<td>30</td>
</tr>
<tr>
<td>WORK</td>
<td>RENTAL_INFO</td>
<td>Title</td>
<td>char</td>
<td>30</td>
</tr>
</tbody>
</table>

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
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:

```sas
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):

![SAS Log Output](image-url)
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:**

```sas
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):**

```sas
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.

<table>
<thead>
<tr>
<th>fmtname</th>
<th>start</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Movie_Rating</td>
<td>G</td>
<td>General Audience</td>
</tr>
<tr>
<td>$Movie_Rating</td>
<td>PG</td>
<td>Parental Guidance</td>
</tr>
<tr>
<td>$Movie_Rating</td>
<td>PG-13</td>
<td>Parental Guidance &gt;= 13</td>
</tr>
<tr>
<td>$Movie_Rating</td>
<td>R</td>
<td>Restricted</td>
</tr>
<tr>
<td>$Movie_Rating</td>
<td>ERROR-Movie_Rating</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Length</th>
<th>Category</th>
<th>Year</th>
<th>Studio</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brave Heart</td>
<td>177</td>
<td>Action Adventure</td>
<td>1996</td>
<td>Paramount Pictures</td>
<td>Restricted</td>
</tr>
<tr>
<td>Casablanca</td>
<td>100</td>
<td>Drama</td>
<td>1942</td>
<td>MGM / UA</td>
<td>Parental Guidance</td>
</tr>
<tr>
<td>Christmas Vacation</td>
<td>97</td>
<td>Comedy</td>
<td>1998</td>
<td>Warner Brothers</td>
<td>Parental Guidance &gt;= 13</td>
</tr>
<tr>
<td>Coming to America</td>
<td>110</td>
<td>Comedy</td>
<td>1988</td>
<td>Paramount Pictures</td>
<td>Restricted</td>
</tr>
<tr>
<td>Dracula</td>
<td>130</td>
<td>Horror</td>
<td>1993</td>
<td>Columbia Tristar</td>
<td>Restricted</td>
</tr>
<tr>
<td>Dressed to Kill</td>
<td>105</td>
<td>Drama Mysteries</td>
<td>1999</td>
<td>Films District</td>
<td>Restricted</td>
</tr>
<tr>
<td>Forrest Gump</td>
<td>142</td>
<td>Drama</td>
<td>1994</td>
<td>Paramount Pictures</td>
<td>Parental Guidance &gt;= 13</td>
</tr>
<tr>
<td>Ghost</td>
<td>127</td>
<td>Drama Romance</td>
<td>1990</td>
<td>Paramount Pictures</td>
<td>Parental Guidance &gt;= 13</td>
</tr>
<tr>
<td>Jeep</td>
<td>125</td>
<td>Action Adventure</td>
<td>1975</td>
<td>Universal Studios</td>
<td>Parental Guidance</td>
</tr>
<tr>
<td>Jurassic Park</td>
<td>127</td>
<td>Action</td>
<td>1994</td>
<td>Universal Pictures</td>
<td>Parental Guidance &gt;= 13</td>
</tr>
<tr>
<td>Lethal Weapon</td>
<td>110</td>
<td>Action Cops &amp; Robber</td>
<td>1987</td>
<td>Warner Brothers</td>
<td>Restricted</td>
</tr>
<tr>
<td>Michael</td>
<td>106</td>
<td>Drama</td>
<td>1987</td>
<td>Warner Brothers</td>
<td>Parental Guidance &gt;= 13</td>
</tr>
<tr>
<td>National Lampoon's Vacation</td>
<td>90</td>
<td>Comedy</td>
<td>1980</td>
<td>Warner Brothers</td>
<td>Parental Guidance &gt;= 13</td>
</tr>
<tr>
<td>Poltergeist</td>
<td>115</td>
<td>Horror</td>
<td>1983</td>
<td>MGM / UA</td>
<td>Parental Guidance</td>
</tr>
<tr>
<td>Rocky</td>
<td>120</td>
<td>Action Adventure</td>
<td>1976</td>
<td>MGM / UA</td>
<td>Parental Guidance</td>
</tr>
<tr>
<td>Scarface</td>
<td>170</td>
<td>Action Cops &amp; Robber</td>
<td>1983</td>
<td>Universal Studios</td>
<td>Restricted</td>
</tr>
<tr>
<td>Silence of the Lambs</td>
<td>118</td>
<td>Drama Suspense</td>
<td>1991</td>
<td>Orion</td>
<td>Restricted</td>
</tr>
<tr>
<td>Star Wars</td>
<td>129</td>
<td>Action Sci-Fi</td>
<td>1977</td>
<td>Lucas Film Ltd</td>
<td>Parental Guidance</td>
</tr>
<tr>
<td>The Hunt for Red October</td>
<td>135</td>
<td>Action Adventure</td>
<td>1989</td>
<td>Paramount Pictures</td>
<td>Parental Guidance</td>
</tr>
<tr>
<td>The Terminator</td>
<td>100</td>
<td>Action Sci-Fi</td>
<td>1984</td>
<td>Live Entertainment</td>
<td>Restricted</td>
</tr>
<tr>
<td>The Wizard of Oz</td>
<td>101</td>
<td>Adventure</td>
<td>1939</td>
<td>MGM / UA</td>
<td>General Audience</td>
</tr>
<tr>
<td>Titanic</td>
<td>194</td>
<td>Drama Romance</td>
<td>1997</td>
<td>Paramount Pictures</td>
<td>Parental Guidance &gt;= 13</td>
</tr>
</tbody>
</table>
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:

```sas
%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;
  %end;
%macro end;
```

---

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):**

```sas
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

Results (4 Excel spreadsheets are produced):

![G-Rated Movies](image1)

![PG-Rated Movies](image2)

![PG-13-Rated Movies](image3)

![R-Rated Movies](image4)
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


Acknowledgments
The author thanks John Cohen and Chuck Kincaid, Building Blocks Section Chairs, for accepting my abstract and paper; Linda Sullivan, SESUG 2018 Academic Chair; Charlotte Baker, SESUG 2018 Operations Chair; the South East SAS Users Group (SESUG) 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