Paper TT-05-2015

SAS® Enterprise Guide® Base SAS® Program Nodes ~ Automating Your SAS World With a Dynamic FILENAME Statement, Dynamic Code, and the CALL EXECUTE Command; *Your Newest BFF (Best Friends Forever) in SAS*

Kent ♥ Ronda Team Phelps, The SASketeers, Des Moines, IA
All for SAS and SAS for All

ABSTRACT

Communication is the basic foundation of all relationships including our relationship with SAS and the Server, PC, or Mainframe. One way to communicate more efficiently and to increasingly automate your SAS World, is to transform Static Code into Dynamic Code that automatically recreates the Static Code and to then execute the Static Code automatically.

Our presentation highlights the powerful SAS Partnership which occurs when a Dynamic FILENAME Statement, Dynamic Code, and the CALL EXECUTE Command are creatively combined within SAS Enterprise Guide Base SAS Program Nodes. You will have the exciting opportunity to learn how **1,469** time-consuming Manual Steps are amazingly replaced with only **2** time-saving Dynamic Automated Steps.

We invite you to attend our session where we will detail the UNIX syntax for our project example and introduce you to your newest BFF (Best Friends Forever) in SAS.

(Please see the Appendices to review starting point information regarding the syntax for Windows and z/OS, and to review the code that created the data sets for our project example.)

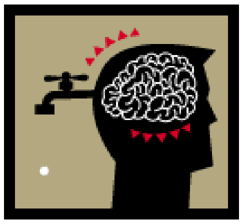
INTRODUCTION

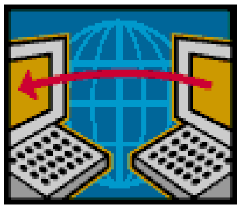
SAS is highly regarded around the world, and rightly so, as a powerful, intuitive, and flexible programming language. As we like to say, SAS enables you to creatively program **S**marter **A**nd **S**marter. However, SAS, as remarkable as it is, is not an island unto itself.

The tagline for SAS is *The Power To Know*® and your 'power to know' greatly expands with your ability to communicate with the Server, PC, or Mainframe (referred to as **server** going forward). **The Power To Know** enables **The Power To Transform** which leads to **The Power To Execute**. However, this power will quickly go down the drain if you do not know how to communicate more efficiently with the server.

Here are 3 questions to ask yourself when designing your SAS program:

- How do I efficiently request data from the server while protecting the integrity of the data?
- How do I automate my program to eliminate time-consuming and error prone manual processing to gain back valuable time for more enjoyable SAS endeavors?
- How do I pursue and accomplish this grand and noble deed?





Good News ~ we are going to show you how to design Base SAS Program Nodes which:

- Transform a Static FILENAME Statement into a Dynamic FILENAME Statement to obtain a Directory Listing of a date range of files from the server.
- Utilize the Directory Listing to transform Extract, Append, and Export Static Code into Dynamic Code
  - Dynamic Code is executable code based upon parameters that can change, and therefore may or may not run exactly the same way.
  - Dynamic Code recreates the Static Code which is executable code that never changes and always runs exactly the same way.
  - Dynamic Code stores the Static Code in a variable in a SAS dataset.
- Run the Dynamic Code and Static Code automatically with no manual processing or intervention.

The SAS project in this presentation demonstrates:

The Power To Know through a Dynamic FILENAME Statement

The Power To Transform Static Code into Dynamic Code that automatically recreates the Static Code

The Power To Execute the recreated Static Code automatically using the CALL EXECUTE Command

We invite you to journey with us as we share how a
Dynamic FILENAME Statement, Dynamic Code, and the CALL EXECUTE Command
became
Best Friends Forever.

☺ A Tale of SAS Wis-h-dom ☺

As stated before, the SAS programming language is powerful, intuitive, and flexible. When we **wish** for a better way to design our programs, we can tap into the built-in **wisdom** of SAS. Thus, we have coined the phrase **SAS Wis-h-dom** to describe the blending of a SAS Wish with SAS Wisdom.

Discovering the power of combining a Dynamic FILENAME Statement, Dynamic Code, and the CALL EXECUTE Command **was**, as Bob Ross, the well-known painter on PBS, so often said, **"A happy accident."** When Bob needed to change his plan for a painting, he referred to the detour as a Happy Accident. Likewise, as we start a project with one plan in mind, we may need to change direction, to persevere in overcoming obstacles, and to discover creative new ways to accomplish the Project Requirements.

On a recent **SAS Quest**, we made several discoveries which we are eager to share with you through our project example. Read on to learn about the Project Requirements, the SAS Wis-h-dom that transpired along the way, and the Happy Accidents which occurred on the journey. This project was prompted by a business need to make the research and analysis of vital variables from **13** years of weekly snapshot data sets more efficient.

Project Requirements:

- **Extract** vital variables from **52** weekly snapshot data sets per year for **13** years **(2003-15)** and combine them with a Load_Date variable (created from the Friday date value derived from the filenames of the data sets) to create **676** new data sets.
- **Append** the **52** new data sets per year to create **13** yearly data sets.
- **Export** the **13** appended yearly data sets back to the folder on the server where the weekly snapshot data sets are stored.

Since SAS Enterprise Guide was being used to design this project, the first decision to make was, “Should the program be designed with Graphical User Interface (GUI) and/or Base SAS Program Nodes?

Here are the questions considered in the programming decision “To GUI or not to GUI?”

- What will it take to **manually** add **52** weekly data sets to the project?
- What will it take to **manually** create **52** queries to select vital variables from **52** data sets?
- What will it take to **manually** enter the derived value of the Load_Date variable in **52** queries?
- What will it take to **manually** append the **52** new data sets created by the **52** queries?
- What will it take to **manually** export the appended yearly data set back to the server?
- Once the program is designed, what will it take to **manually** swap **52** inputs and **manually** update the Load_Date variable in **52** queries – **12** more times – while running the program for the **13** year timeframe?

Are you getting tired yet?

It was determined that the **209 manual steps** needed to design the program, and the **105 manual steps** needed to update the program each year, could be done with GUI. However, it also became apparent that the **1,469 manual steps** required to run the program for the **13** year timeframe would be excessive and prone to errors. As a result, our **SAS Intuition** said, “There must be a smarter, easier, and faster way to do this in SAS!”

Dear SAS Wisdom,
We wish we could find a way to automate this program
and eliminate manual processing and intervention –
except of course for choosing the year.
We look forward to hearing from you soon,
Thank You

By the way, are you in tune with your SAS Intuition? Be sure to listen closely when the quiet, reassuring voice within you says with conviction, “There must be a better way to do this in SAS!” We encourage you to honor your SAS Intuition and to let it motivate you to find new ways to maximize your programming.

“And now for the rest of the story…”,
as Paul Harvey so often said on the radio.

The SAS Quest

Starting
is the first step
towards success.
John C. Maxwell

Sometimes at the beginning of a project it can be challenging to figure out how to accomplish the requirements. Always remember, the only thing we really need to do is take the first step ~ and the rest will soon follow.

☺ Team Phelps Law ☺

Everything is easier than it looks;
it will be more rewarding than you expect;
and if anything can go right
~ it will ~
and at the best possible moment.

Our first step was to revise the previous programming questions:

- What will it take to **automatically** create **52** DATA steps to read **52** data sets?
- What will it take to **automatically** extract vital variables in **52** DATA steps?
- What will it take to **automatically** enter the derived value of the Load_Date variable in **52** DATA steps?
- What will it take to **automatically** append the **52** new data sets created by the **52** DATA steps?
- What will it take to **automatically** export the appended yearly data set back to the server?
- Once the program is designed, what will it take to **automatically** swap **52** inputs and **automatically** update the Load_Date variable in **52** DATA steps – **12** more times – while running the program for the **13** year timeframe?

We decided to automate this program and began a quest to accomplish this grand and noble deed ☺. The next step was to find a way to transform a Static FILENAME Statement into a Dynamic FILENAME Statement to read **52** weekly data sets from a folder on the server automatically and sequentially ~ rather than manually one at a time. A Google search led to an article titled *Using FILEVAR= To Read Multiple External Files in a DATA Step*.

Here is a brief overview of the article:

- The article explained different ways to create and use a Dynamic FILENAME Statement to automatically and sequentially read the content of multiple files.
- Unfortunately, the examples seemed to indicate that they cannot also derive the value of a variable from the filenames of the files being read ~ Bummer!

☺ **Happy Accident Alert** ☺ ~ We kept searching and discovered that when a FILENAME Statement is used, SAS will assign a variable called FILENAME to the name of each file being read ~ Yea! We can use a Dynamic FILENAME Statement to obtain a Directory Listing of the filenames which can then be utilized to read the content of the files while also deriving the value of a variable from the filenames of the files being read.

Learning this information initiated 2 programs:

- ❖ **Program 1 ~** Design a Dynamic FILENAME Statement to obtain one Directory Listing (per year for **13** years of the filenames of the **52** weekly snapshot data sets), and utilize the Directory Listing to transform **Extract Static Code** into Dynamic Code that automatically recreates the Static Code to Extract vital variables (from the data sets) and combine them with a Load_Date variable (created from the Friday date value derived from the filenames of the data sets) to create **52** new data sets per year.
- ❖ **Program 2 ~** Utilize the Directory Listing to transform **Append and Export Static Code** into Dynamic Code that automatically recreates the Static Code to Append the **52** new data sets per year to create **13** yearly data sets, and to Export the **13** appended yearly data sets back to the folder on the server where the weekly snapshot data sets are stored.

Once the **2** programs are run, the recreated Extract, Append, and Export Static Code can be run **manually** by copying and pasting the code into another Program Node. These **2** programs fulfill most of the project requirements… but remember, our SAS Wish was to **COMPLETELY** automate this project.

As we continue on our journey, we will shed more light on this exciting SAS Quest.

SAS Illumination

Sometimes success is seeing
what we already have
in a new light.

Dan Miller

After we determined how to transform a Static FILENAME Statement into a Dynamic FILENAME Statement to obtain a Directory Listing to utilize in transforming Extract, Append, and Export Static Code into Dynamic Code that automatically recreates the Static Code; a very important question arose, "Is there also a way to execute the Static Code automatically?" SAS Intuition spoke again, "There must be a way to call and execute a variable in a SAS data set containing a SAS DATA step."

☺ **Happy Accident Alert** ☺ **~** Another Google search led to a White Paper titled *CALL EXECUTE: A Powerful Data Management Tool* which unexpectedly revealed that a CALL EXECUTE command already existed!

Here is a brief overview of the White Paper:

- ❖ CALL EXECUTE (variable); resolves and executes the value of a variable.
- ❖ The variable can be a character variable in a data set containing SAS statements such as a DATA step.
- ❖ The CALL EXECUTE Command will execute the recreated Static Code automatically and will enable us to finally fulfill all of the project requirements!

Yea!!!

Strike up the band,

Toss the confetti,

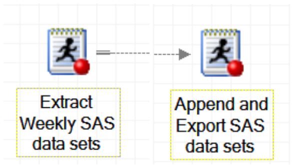
Release the balloons!

Applause… Applause… Applause…

Bring out the treats for everyone!

SAS Illumination ~ We will use a Dynamic FILENAME Statement to obtain a Directory Listing to utilize in transforming Extract, Append, and Export Static Code into Dynamic Code that automatically recreates the Static Code and then use the CALL EXECUTE Command to execute the Static Code automatically. The programs will run automatically without any manual processing or intervention ~ except for choosing the year!

Here are the 2 programs displayed as SAS Enterprise Guide Base SAS Program Nodes:



As you can see from this SAS Quest, it pays to listen to your SAS Intuition. Simple Google searches led to resources which illuminated how to fulfill the project requirements and enabled this project to become a very successful reality. Always remember the treasure trove of SAS information waiting on the web to help you maximize the quality and efficiency of your programming.

On the next leg of our journey

we will walk you through a

step-by-step demonstration of

The Power To Know, Transform, and Execute.

The first step is the most important step you will take,
And the last step is the most rewarding step you will experience.

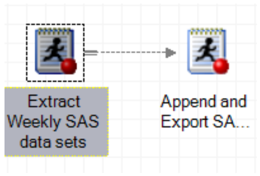
Kent ♥ Ronda Team Phelps

THE POWER TO KNOW
Through a Dynamic FILENAME Statement

Disclaimer: This presentation details the UNIX syntax for our project example. Please refer to your specific Operating System (e.g. UNIX, Windows, or z/OS) Manual, Installation Configuration, and/or in-house Technical Support for further guidance in how to create the SAS code presented in this paper. Please see Appendix A for starting point information regarding the syntax for Windows and z/OS.

The following examples highlight how to transform a Static FILENAME Statement into a Dynamic FILENAME Statement to obtain a Directory Listing of the filenames of the **52** weekly data sets for the year **2015** from a folder on the server.

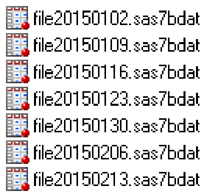
This code will obtain and store the Directory Listing:



```
FILENAME indata '/data/MWSUG/CALL_EXECUTE/file2015*.sas7bdat';
DATA path_list_files;
    LENGTH fpath SAS_data_set_and_path $100;
    RETAIN fpath;
    INFILE indata TRUNCOVER FILENAME=SAS_data_set_and_path;
    INPUT;
    IF fpath NE SAS_data_set_and_path THEN;
        DO;
            fpath = SAS_data_set_and_path;
            . . .
            OUTPUT;
        END;
RUN;
```

❖ This code will obtain a Directory Listing of the data sets following the `file2015*.sas7bdat` pattern from the `/data/MWSUG/CALL_EXECUTE` folder on the server and store it in a data set.

These are the 7 weekly data sets being processed in our example:

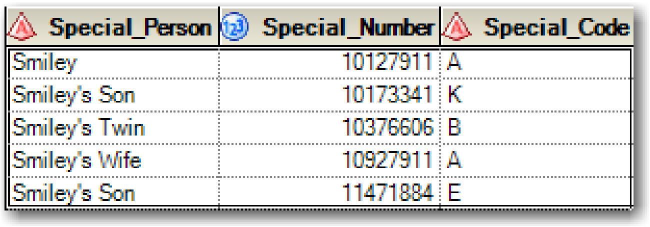


file20150102.sas7bdat
file20150109.sas7bdat
file20150116.sas7bdat
file20150123.sas7bdat
file20150130.sas7bdat
file20150206.sas7bdat
file20150213.sas7bdat

❖ Each of these data sets must follow the same pattern of **fileYYYYMMDD.sas7bdat**.

❖ Please see **Appendix B** for the code that creates these data sets.

This is a sample of the columns and formatting for each data set:



Special_Person	Special_Number	Special_Code
Smiley	10127911	A
Smiley's Son	10173341	K
Smiley's Twin	10376606	B
Smiley's Wife	10927911	A
Smiley's Son	11471884	E

- The data sets contain each Special Person, Special Number, and Special Code for the employees of the ☺ **Smiley Company** ☺.

Creating a Dynamic FILENAME Statement:

```
FILENAME indata '/data/MWSUG/CALL_EXECUTE/file2015*.sas7bdat';
```

- The **FILENAME** Statement assigns **indata** as a file reference (fileref) to the folder and file pattern.
- The asterisk within the file pattern **file2015*.sas7bdat** transforms the Static **FILENAME** Statement into a Dynamic **FILENAME** Statement which will read multiple files automatically and sequentially.
- The **FILENAME=variable** Statement assigns the path and name of each file being read.
- In summary, a Dynamic **FILENAME** Statement and the **FILENAME=variable** Statement will obtain the Directory Listing.

Creating a DATA step which will read and store the Directory Listing:

```
DATA path_list_files;
     LENGTH fpath SAS_data_set_and_path $100;
     RETAIN fpath;
```

- The **DATA** Statement creates an output data set called **path_list_files**.
- The **LENGTH** Statement assigns a length of 100 characters to a variable that will store each unique data set path and filename called **fpath**.
- The **RETAIN** Statement retains the value of **fpath** until it is assigned a new filename later in the code.
- The **LENGTH** Statement also assigns a length of 100 characters to a variable that will be used to store and track changes to the data set path and filename called **SAS_data_set_and_path**.
- In summary, the **path_list_files** data set is created to contain the 100 character **fpath** and **SAS_data_set_and_path** variables which will be used to read and store the Directory Listing.

Preparing the INFILE indata (fileref) for use and the INPUT of data:

```
INFILE indata TRUNCOVER FILENAME=SAS_data_set_and_path;
INPUT;
```

- ❖ The `INFILE` Statement assigns `indata` to be read with the `INPUT` Statement.
- ❖ The `TRUNCOVER` option tells SAS the input data may or may not be the same length.
- ❖ The `FILENAME=SAS_data_set_and_path` Statement assigns `SAS_data_set_and_path` to the path and filename of the file being read.
- ❖ The `INPUT` Statement reads the `INFILE` `indata` (fileref) sequentially without creating any variables.
- ❖ In summary, `INFILE` assigns `indata` to be read with an `INPUT` of variable length (without creating any variables) while assigning `SAS_data_set_and_path` to the filename of each file being read.

Creating an IF-THEN DO-END Statement to detect new filenames being read:

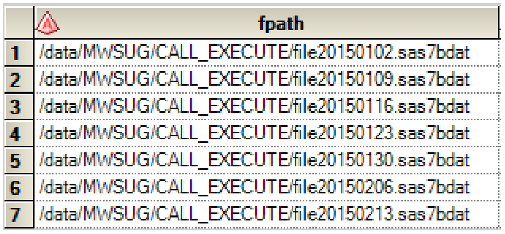
```
IF fpath NE SAS_data_set_and_path THEN;
    DO;
        fpath = SAS_data_set_and_path;
        ...
        OUTPUT;
    END;
```

- ❖ The `IF-THEN` Statement executes the contents of the `DO-END` when a new filename is read.
- ❖ The `fpath = SAS_data_set_and_path` Statement assigns the `fpath` variable to the value of the `SAS_data_set_and_path` variable which contains the path and filename as each new file is read.
- ❖ The `OUTPUT` Statement is executed within the `IF-THEN` `DO-END` Statement to ensure that we only write an observation recreating Static Code when a new file is read and `fpath` changes.
- ❖ In summary, the `fpath` variable is assigned to the path and filename of each new data set (Directory Listing) up to `100` characters as the filename of the data sets change.

Here are the statements combined with a RUN Statement:

```
FILENAME indata '/data/MWSUG/CALL_EXECUTE/file2015*.sas7bdat';
DATA path_list_files;
    LENGTH fpath SAS_data_set_and_path $100;
    RETAIN fpath;
    INFILE indata TRUNCOVER FILENAME=SAS_data_set_and_path;
    INPUT;
    IF fpath NE SAS_data_set_and_path THEN;
        DO;
            fpath = SAS_data_set_and_path;
            ...
            OUTPUT;
        END;
RUN;
```

This is the output data set created by the preceding statements:



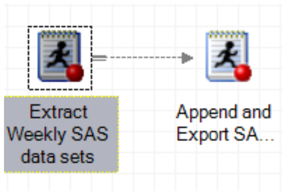
	fpath
1	/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat
2	/data/MWSUG/CALL_EXECUTE/file20150109.sas7bdat
3	/data/MWSUG/CALL_EXECUTE/file20150116.sas7bdat
4	/data/MWSUG/CALL_EXECUTE/file20150123.sas7bdat
5	/data/MWSUG/CALL_EXECUTE/file20150130.sas7bdat
6	/data/MWSUG/CALL_EXECUTE/file20150206.sas7bdat
7	/data/MWSUG/CALL_EXECUTE/file20150213.sas7bdat

❖ In the next section we will explore how the **`fpath`** variable is used to transform Static Code into Dynamic Code.

THE POWER TO TRANSFORM
Static Code Into Dynamic Code

The following examples highlight how to transform **Extract Static Code** into Dynamic Code that automatically recreates the Static Code to Extract vital variables from **52** weekly data sets and combine them with a Load_Date variable (created from the Friday date value derived from the filenames of the data sets) to create **52** new data sets.

This is the original Extract Static Code:



```
DATA file_final_20150102;
    SET '/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat';
    FORMAT Load_Date date9.; Load_Date = '02JAN2015'd;
    KEEP Special_Person Special_Number Load_Date;
RUN;
```

How to transform Extract Static Code into Dynamic Code:

```
FILENAME indata '/data/MWSUG/CALL_EXECUTE/file2015*.sas7bdat';
DATA path_list_files;
    LENGTH SAS_data_set_and_path fpath $100 fpath_line $1000;
    RETAIN fpath;
    FORMAT Load_Date date9.;
    INFILE indata TRUNCOVER FILENAME=SAS_data_set_and_path;
    INPUT;
    IF fpath NE SAS_data_set_and_path THEN
        DO;
            fpath = SAS_data_set_and_path;
fpath_line = CATS("DATA file_final_20150102;
                       SET '/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat';
                       FORMAT Load_Date date9.;
                       Load_Date = '02JAN2015'd;
                       KEEP Special_Person Special_Number Load_Date;
                   RUN;");
            OUTPUT;
        END;
RUN;
```

- Surround the Static Code with quotation marks to begin the process of transforming the code.
- If single quotes are contained within the Static Code, use double quotes to surround the Static Code.
- Create a variable **fpath_line** that is assigned to the concatenation with spaces removed (**CATS**) of the Static Code in quotation marks.

Identify what changes with each observation of Static Code:

```
fpath_line = CATS("DATA file_final_","20150102",";
                  SET '","/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat","';
                  FORMAT Load_Date date9.;
                  Load_Date = '","02JAN2015","'d;
                  KEEP Special_Person Special_Number Load_Date;
                  RUN;");
```

- ❖ The dates and the name of the input file will change with each observation, so surround this code with double quotes.

Replace what changes with each observation of Static Code:

```
      Load_Date_Text = SUBSTR(fpath,30,8);
      fpath_line = CATS("DATA file_final_",Load_Date_Text,";
                            SET '",fpath,"';
                            FORMAT Load_Date date9.;
                            Load_Date = '","02JAN2015","'d;
                            KEEP Special_Person Special_Number Load_Date;
                        RUN;");
```

- ❖ Replace the "20150102" with a **Load_Date_Text** variable derived from the value of the load date of the output file from the **fpath** variable containing the path and name of each input file.
- ❖ Replace the path and filename with the **fpath** variable for each input file.

This Dynamic Code automatically recreates the Static Code:

```
FILENAME indata '/data/MWSUG/CALL_EXECUTE/file2015*.sas7bdat';
DATA path_list_files;
       LENGTH SAS_data_set_and_path fpath $100 fpath_line $1000;
       RETAIN fpath;
    FORMAT Load_Date date9.;
    INFILE indata TRUNCOVER FILENAME=SAS_data_set_and_path;
    INPUT;
    IF fpath NE SAS_data_set_and_path THEN
        DO;
            fpath = SAS_data_set_and_path;
            Load_Date_Text = SUBSTR(fpath,30,8);
            Load_Date = MDY(INPUT(SUBSTR(Load_Date_Text,5,2),2.),
                            INPUT(SUBSTR(Load_Date_Text,7,2),2.),
                            INPUT(SUBSTR(Load_Date_Text,1,4),4.));
        fpath_line = CATS("DATA file_final_",Load_Date_Text,";
                             SET '",fpath,"';
                             FORMAT Load_Date date9.;
                             Load_Date = '",PUT(Load_Date,date9.),"'d;
                             KEEP Special_Person Special_Number Load_Date;
                         RUN;");
            OUTPUT;
        END;
RUN;
```

- ❖ Derive the `Load_Date` variable with the MDY function using the **Load_Date_Text** variable.
- ❖ Replace the "02JAN2015" with the **PUT(Load_Date,date9.)** to create a character SAS date.
- ❖ Next we will focus on resolving the Dynamic Code to show how it recreates the Static Code.

Here is how we resolve the Dynamic Code starting with Load_Date_Text:

```
Load_Date_Text = SUBSTR(fpath,30,8);
Load_Date = MDY(INPUT(SUBSTR(Load_Date_Text,5,2),2.),
                INPUT(SUBSTR(Load_Date_Text,7,2),2.),
                INPUT(SUBSTR(Load_Date_Text,1,4),4.));
```

❖ The **fpath** variable contains the path and filename of each data set in the following format:

```
/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat  (fpath contents – 1st observation)
12345678901234567890123456789012345678901234567890  (character spacing)
```

❖ The **SUBSTR** function sets **Load_Date_Text** to **'20150102'** – begins with character 30 of **fpath** for 8 characters.

❖ The **SUBSTR** function obtains the month **' 01'**, day **'02'**, and year **'2015'** from **Load_Date_Text**:

```
Load_Date = MDY(INPUT(SUBSTR('20150102',5,2),2.),
                INPUT(SUBSTR('20150102',7,2),2.),
                INPUT(SUBSTR('20150102',1,4),4.));
```

❖ The **INPUT** function converts the character values of month, day, and year to numeric values:

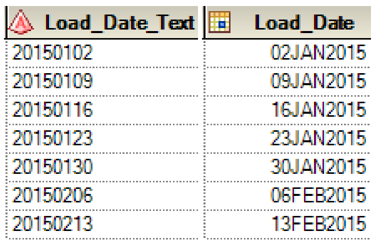
```
Load_Date = MDY(INPUT('01',2.),INPUT('02',2.),INPUT('2015',4.));
```

❖ The **MDY** function converts the numeric values of month, day, and year to a SAS date:

```
Load_Date = MDY(1,2,2015);
```

❖ Since Load_Date was formatted as date9 by the earlier **FORMAT** Statement, this resolves to:

```
Load_Date = '02JAN2015'd;
```



Load_Date_Text	Load_Date
20150102	02JAN2015
20150109	09JAN2015
20150116	16JAN2015
20150123	23JAN2015
20150130	30JAN2015
20150206	06FEB2015
20150213	13FEB2015

Once Load_Date is assigned, Load_Date_Text, fpath, and Load_Date are used to create the 1st set of Static Code:

```
     fpath_line = CATS("DATA file_final_",Load_Date_Text,";
                           SET '",fpath,"';
                           FORMAT Load_Date date9.;
                           Load_Date = '",PUT(Load_Date,date9.),"'d;
                           KEEP Special_Person Special_Number Load_Date;
                       RUN;");
```

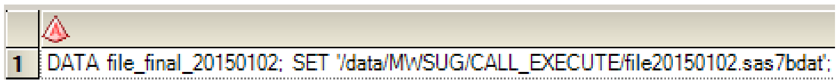
- The **PUT** function is used to convert the **Load_Date** from a numeric SAS date to a character SAS date.
- The **Load_Date_Text**, **fpath**, and **Load_Date** variables resolve to:

```
fpath_line = CATS('DATA file_final_',20150102,
                    "; SET '","/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat,
                   "'; FORMAT Load_Date date9.; Load_Date =",'02JAN2015'd;",
                     ' KEEP Special_Person Special_Number Load_Date;
                   RUN;');
```

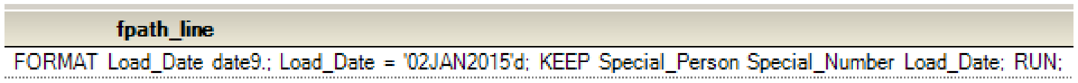
- The **CATS** function concatenates what is separated by commas while removing leading and trailing spaces:

```
   fpath_line = "DATA file_final_20150102;
                      SET '/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat';
                      FORMAT Load_Date date9.; Load_Date = '02JAN2015'd;
                      KEEP Special_Person Special_Number Load_Date;
                  RUN;";
```

- You may be asking yourself, "Why do the **FORMAT** Satement and the **Load_Date** assignment appear here since they were already included in the code discussed earlier?"
- Good question – remember, this Static Code will run apart from the Dynamic Code, so the Static Code needs to be self-contained with all of the statements and syntax necessary to run on its own.
- The **KEEP** Statement enables you to create the output data set with only the vital variables listed:

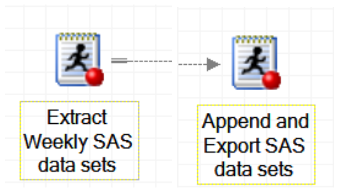


```
1  DATA file_final_20150102; SET '/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat';
```



```
fpath_line
FORMAT Load_Date date9.; Load_Date = '02JAN2015'd; KEEP Special_Person Special_Number Load_Date; RUN;
```

Here are the 2 programs displayed as Base SAS Program Nodes:



Here is 1 observation of the Static Code created by the 2 programs:

SAS Dataset: `PATH_FILE_LIST`
Variable: `fpath_line`

```
DATA file_final_20150102;
    SET '/data/MWSUG/CALL_EXECUTE/
        file20150102.sas7bdat';
    FORMAT Load_Date date9.;
    Load_Date = '02JAN2015'd;
    KEEP Special_Person
         Special_Number
         Load_Date;
RUN;
```

To

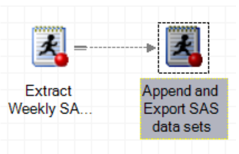
Be

Illuminated

The first part of **THE POWER TO TRANSFORM** section has walked us through the process of transforming **Extract Static Code** into Dynamic Code that automatically recreates the Static Code to Extract vital variables from **52** weekly data sets and combine them with a Load_Date variable (created from the Friday date value derived from the filenames of the data sets) to create **52** new data sets. The Extract Static Code is contained in the **fpath_line** variable.

The following examples highlight how to transform **Append and Export Static Code** into Dynamic Code that automatically recreates the Static Code to Append the 52 new data sets to create a yearly data set, and to Export the yearly data set back to the server.

This is the original Append and Export Static Code:



```
DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015';
    SET file_final_20150102
        file_final_20150109
        file_final_20150116
        file_final_20150123
        file_final_20150130
        file_final_20150206
        file_final_20150213;
RUN;
```

How to transform Append and Export Static Code into Dynamic Code:

```
DATA prepare_historical_append;
    SET path_list_files END=LAST_OBS;
    LENGTH append_and_export $2000;
    RETAIN append_and_export;
    KEEP append_and_export;

    append_and_export = CATS("
        DATA '/data/MWSUG/CALL_EXECUTE/file_all_2015';
            SET file_final_20150102
                file_final_20150109
                file_final_20150116
                file_final_20150123
                file_final_20150130
                file_final_20150206
                file_final_20150213;
        RUN;");
RUN;
```

❖ Surround the Static Code with quotation marks to begin the process of transforming the code.

❖ If single quotes are contained within the Static Code, use double quotes to surround the Static Code.

❖ Create a variable **append_and_export** that is assigned to the concatenation with spaces removed (**CATS**) of the Static Code in quotation marks.

❖ The **RETAIN** statement will retain the value of **append_and_export** as new input files are added with each observation.

Identify what changes with each observation of Static Code:

```
            append_and_export = CATS("
                DATA '/data/MWSUG/CALL_EXECUTE/file_all_","2015","';
                    SET file_final_","20150102","
                        file_final_","20150109","
                        file_final_","20150116","
                        file_final_","20150123","
                        file_final_","20150130","
                        file_final_","20150206","
                        file_final_","20150213",";
                RUN;");
```

❖ The dates and the name of the input file will change with each observation, so surround this code with double quotes.

Replace what changes with each observation of Static Code:

```
append_and_export = CATS("
    DATA '/data/MWSUG/CALL_EXECUTE/file_all_",SUBSTR(Load_Date_Text,1,4),"';
        SET "," file_final_","20150102","
                file_final_","20150109","
                file_final_","20150116","
                file_final_","20150123","
                file_final_","20150130","
                file_final_","20150206","
                file_final_","20150213",";
    RUN;");
```

- ❖ Replace the **"2015"** with **SUBSTR(Load_Date_Text,1,4)** to derive the name of the output file.
- ❖ The **DATA** step line date variable is static once it is set to the year.
- ❖ All of the input files that are **SET** will change with every observation.
- ❖ The **RUN** line is static once all of the other Static Code has been derived.

This Dynamic Code automatically recreates the Static Code:

```
DATA prepare_historical_append;
    SET path_list_files END=LAST_OBS;
    LENGTH append_and_export $2000;
    RETAIN append_and_export;
    KEEP append_and_export;
    IF _N_ = 1 THEN append_and_export =
        "DATA '/data/MWSUG/CALL_EXECUTE/file_all_"||
            SUBSTR(Load_Date_Text,1,4)||"'; SET ";

    append_and_export = CATS(append_and_export)||
        " file_final_"||Load_Date_Text;

    IF LAST_OBS THEN
        DO;
            append_and_export = CATS(append_and_export)||"; RUN;";
            OUTPUT;
        END;
RUN;
```

- ❖ To create the **DATA** step and **SET** only once, they are assigned with the first observation (**_N_ = 1**).
- ❖ Then with every observation, each input file is assigned and concatenated to **append_and_export**.
- ❖ **CATS** is used to remove spaces but **||** is used to keep the space before each input file.
- ❖ **LAST_OBS** is used to concatenate **RUN** at the end of the input files and to **OUTPUT** a single observation.
- ❖ Next we will focus on resolving the Dynamic Code to show how it recreates the Static Code.

Here is how we resolve the Dynamic Code starting with the IF _N_ = 1:

```
IF _N_ = 1 THEN append_and_export =
   "DATA '/data/MWSUG/CALL_EXECUTE/file_all_"||
      SUBSTR(Load_Date_Text,1,4)||"'; SET ";
```

❖ The `IF-THEN` statement only executes while processing the first input observation `_N_ = 1`.

❖ The `Load_Date_Text` variable resolves to:

```
append_and_export = "DATA '/data/MWSUG/CALL_EXECUTE/file_all_"||
                    SUBSTR(Load_Date_Text,1,4)||"'; SET ";
```

❖ The `SUBSTR` function resolves to '2015':

```
append_and_export = "DATA '/data/MWSUG/CALL_EXECUTE/file_all_"||'2015'||"'; SET ";
```

❖ The `||` (concatenation without removing spaces) resolves to:

```
append_and_export = "DATA '/data/MWSUG/CALL_EXECUTE/file_all_2015'; SET ";
```

❖ Notice how `append_and_export` looks like the beginning of a **DATA** step.

The history_append_line variable is then derived from itself and Load_Date_Text again for all observations:

```
append_and_export = CATS(append_and_export)||" file_final_"||Load_Date_Text;
```

❖ The `append_and_export` and `Load_Date_Text` variables resolve to:

```
append_and_export = CATS("DATA '/data/MWSUG/CALL_EXECUTE/file_all_2015'; SET ")||
                    " file_final_"||'20150102';
```

❖ The `CATS` and the `||` resolve to:

```
append_and_export = "DATA '/data/MWSUG/CALL_EXECUTE/file_all_2015';
                     SET file_final_20150102";
```

The history_append_line variable continues to be derived from itself and Load_Date_Text for all observations:

```
append_and_export = CATS(append_and_export)||" file_final_"||Load_Date_Text;
```

❖ The `append_and_export` always resolves to the way it looked at the end of the previous assignment statement and then concatenates with the name of the next file:

```
append_and_export = "DATA '/data/MWSUG/CALL_EXECUTE/file_all_2015';
                     SET file_final_20150102
                         file_final_20150109
                         ...";
```

❖ This will continue until the last Filename is added:

```
append_and_export = "DATA '/data/MWSUG/CALL_EXECUTE/file_all_2015';
                     SET file_final_20150102
                         ...
                         file_final_20150213";
```

The append_and_export variable is then derived from itself and Load_Date_Text a final time for the last observation:

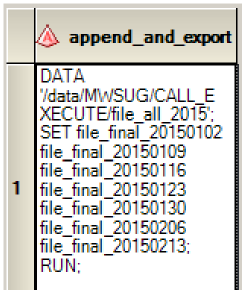
```
IF LAST_OBS THEN
    DO;
        append_and_export = CATS(append_and_export)||"; RUN;";
        OUTPUT;
    END;
```

❖ Once the last observation is read, the `append_and_export` always resolves to the way it looked at the end of the previous assignment statement and then concatenates with `'; RUN;'`:

```
append_and_export = "DATA '/data/MWSUG/CALL_EXECUTE/file_all_2015';
                     SET file_final_20150102
                         ...
                         file_final_20150213;
                 RUN;";
```

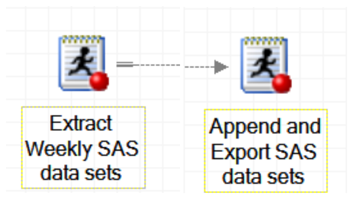
❖ The `OUTPUT` Statement is executed within `IF-THEN DO-END` because only one observation is needed in the output data set containing the completed `append_and_export`.

❖ Here is how the **only** observation appears in the `prepare_historical_append` data set:



	append_and_export
1	DATA '/data/MWSUG/CALL_EXECUTE/file_all_2015'; SET file_final_20150102 file_final_20150109 file_final_20150116 file_final_20150123 file_final_20150130 file_final_20150206 file_final_20150213; RUN;

Here are the 2 programs displayed as Base SAS Program Nodes:



Here is 1 observation of the Static Code created by the 2 programs:

SAS Dataset: PATH_FILE_LIST
Variable: fpath_line

```
DATA file_final_20150102;
    SET '/data/MWSUG/CALL_EXECUTE/
        file20150102.sas7bdat';
    FORMAT Load_Date date9.;
    Load_Date = '02JAN2015'd;
    KEEP Special_Person
         Special_Number
         Load_Date;
RUN;
```

SAS Dataset:
prepare_historical_append
Variable: append_and_export

```
DATA '/data/MWSUG/CALL_EXECUTE/
      file_final_2015';
 SET file_final_20150102
     file_final_20150109
     file_final_20150116
     file_final_20150123
     file_final_20150230
     file_final_20150207
     file_final_20150213;
RUN;
```

The second part of **THE POWER TO TRANSFORM** section has walked us through the process of transforming **Append and Export Static Code** into Dynamic Code that automatically recreates the Static Code to Append the **52** new data sets to create a yearly data set, and to Export the yearly data set back to the server. The Append and Export Static Code is contained in the **append_and_export** variable.

THE POWER TO EXECUTE
Static Code Automatically Using The CALL EXECUTE Command

After we transform the Static Code into Dynamic Code that automatically recreates the Static Code to **Extract, Append, and Export** the appended yearly data set, the CALL EXECUTE Command is used to execute the **2** sets of Static Code automatically.

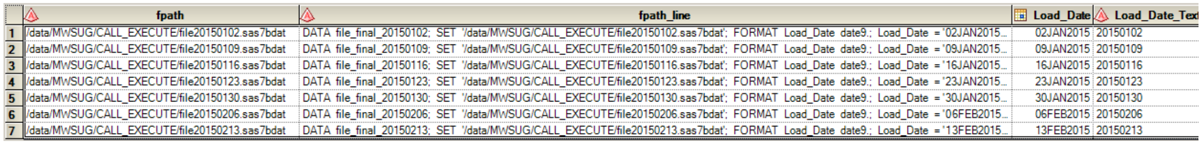
Executing the Extract Static Code using the CALL EXECUTE Command:

```
DATA _NULL_;
    SET path_list_files;
    CALL EXECUTE(fpath_line);
RUN;
```

Creating a DATA step that executes Static Code to Extract data sets:

```
DATA _NULL_;
    SET path_list_files;
```

- The **CALL EXECUTE** Statement does not create an output data set because the **_NULL_** option is used.
- The **SET** Statement sets **path_list_files** as the input data set for this **DATA** step.
- Here are the **7** observations in the **path_list_files** data set:



	fpath	fpath_line	Load_Date	Load_Date_Tex
1	/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat	DATA file_final_20150102; SET '/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat'; FORMAT Load_Date date9.; Load_Date = '02JAN2015...	02JAN2015	20150102
2	/data/MWSUG/CALL_EXECUTE/file20150109.sas7bdat	DATA file_final_20150109; SET '/data/MWSUG/CALL_EXECUTE/file20150109.sas7bdat'; FORMAT Load_Date date9.; Load_Date = '09JAN2015...	09JAN2015	20150109
3	/data/MWSUG/CALL_EXECUTE/file20150116.sas7bdat	DATA file_final_20150116; SET '/data/MWSUG/CALL_EXECUTE/file20150116.sas7bdat'; FORMAT Load_Date date9.; Load_Date = '16JAN2015...	16JAN2015	20150116
4	/data/MWSUG/CALL_EXECUTE/file20150123.sas7bdat	DATA file_final_20150123; SET '/data/MWSUG/CALL_EXECUTE/file20150123.sas7bdat'; FORMAT Load_Date date9.; Load_Date = '23JAN2015...	23JAN2015	20150123
5	/data/MWSUG/CALL_EXECUTE/file20150130.sas7bdat	DATA file_final_20150130; SET '/data/MWSUG/CALL_EXECUTE/file20150130.sas7bdat'; FORMAT Load_Date date9.; Load_Date = '30JAN2015...	30JAN2015	20150130
6	/data/MWSUG/CALL_EXECUTE/file20150206.sas7bdat	DATA file_final_20150206; SET '/data/MWSUG/CALL_EXECUTE/file20150206.sas7bdat'; FORMAT Load_Date date9.; Load_Date = '06FEB2015...	06FEB2015	20150206
7	/data/MWSUG/CALL_EXECUTE/file20150213.sas7bdat	DATA file_final_20150213; SET '/data/MWSUG/CALL_EXECUTE/file20150213.sas7bdat'; FORMAT Load_Date date9.; Load_Date = '13FEB2015...	13FEB2015	20150213

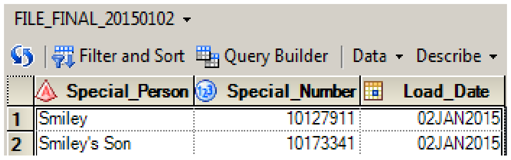
The CALL EXECUTE Command executes the fpath_line variable:

```
                    CALL EXECUTE(fpath_line);
                RUN;
```

❖ The **CALL EXECUTE** Command executes the contents of the **fpath_line** variable in the **path_list_files** data set. Here is the first observation of **fpath_line** in the **path_list_files** data set:

```
DATA file_final_20150102;
    SET '/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat';
    FORMAT Load_Date  date9. Birth_Date date9.;
    Load_Date = '02JAN2015'd;
    KEEP Special_Person Special_Number Load_Date;
RUN;
```

❖ This is the result of executing the first observation of **fpath_line** in the **path_list_files** data set:

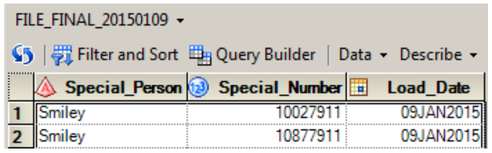


	Special_Person	Special_Number	Load_Date
1	Smiley	10127911	02JAN2015
2	Smiley's Son	10173341	02JAN2015

❖ The **RUN** Statement causes the second observation of **fpath_line** in the **path_list_files** data set to be read:

```
DATA file_final_20150109;
    SET '/data/MWSUG/CALL_EXECUTE/file20150109.sas7bdat';
    FORMAT Load_Date  date9. Birth_Date date9.;
    Load_Date = '09JAN2015'd;
    KEEP Special_Person Special_Number Load_Date;
RUN;
```

❖ This is the result of executing the second observation of **fpath_line** in the **path_list_files** data set:



	Special_Person	Special_Number	Load_Date
1	Smiley	10027911	09JAN2015
2	Smiley	10877911	09JAN2015

❖ The execution of **fpath_line** continues for each observation in the **path_list_files** data set.

Once the Static Code to automatically **Extract** vital variables from the **52** weekly data sets and combine them with a Load_Date variable has been executed, the next step is to execute the **Append and Export Static Code.**

Executing the Append and Export Static Code using the CALL EXECUTE Command:

```
DATA _NULL_;
    SET prepare_historical_append;
    CALL EXECUTE(append_and_export);
RUN;
```

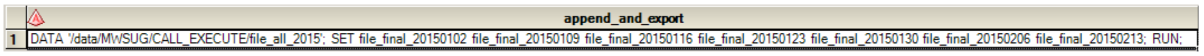
Creating a DATA step that executes Static Code to Append data sets:

```
DATA _NULL_;
    SET prepare_historical_append;
```

❖ The **DATA** Statement does not create an output data set because the **_NULL_** option is used.

❖ The **SET** Statement sets **prepare_historical_append** as the input data set for this **DATA** step.

❖ Here is the **only** observation in the **prepare_historical_append** data set:



	append_and_export
1	DATA '/data/MWSUG/CALL_EXECUTE/file_all_2015'; SET file_final_20150102 file_final_20150109 file_final_20150116 file_final_20150123 file_final_20150130 file_final_20150206 file_final_20150213; RUN;

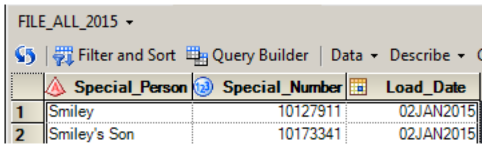
The CALL EXECUTE Command executes the append_and_export variable:

```
    CALL EXECUTE(append_and_export);
RUN;
```

❖ The **CALL EXECUTE** Command executes the contents of the **append_and_export** variable in the **prepare_historical_append** data set. Here is the **only** observation of **append_and_export**:

```
DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015';
    SET file_final_20150102
        file_final_20150109
        file_final_20150116
        file_final_20150123
        file_final_20150230
        file_final_20150206
        file_final_20150213;
RUN;
```

❖ Here is the result of executing **append_and_export** in the **prepare_historical_append** data set:



	Special_Person	Special_Number	Load_Date
1	Smiley	10127911	02JAN2015
2	Smiley's Son	10173341	02JAN2015

Now that we have completed the process for **1** year, we need to repeat the process for the remaining **12** years of this project. How is this accomplished? We simply **update the year** in the variable portion of the Dynamic FILENAME Statement in the **Extract Program Node**, rerun both Program Nodes, and then repeat this process until each of the remaining years is complete.

Creating the yearly data sets for each year:

```
FILENAME indata '/data/MWSUG/CALL_EXECUTE/file2015*.sas7bdat';
```

❖ Update the year in the Extract Program Node and then rerun both Program Nodes for each year.

☺ Done and Done ☺

CONCLUSION

The Power To Know through a Dynamic FILENAME Statement enables **The Power To Transform** Static Code into Dynamic Code that automatically recreates the Static Code and leads to **The Power To Execute** the recreated Static Code automatically using the CALL EXECUTE Command ~ ☺ try saying that statement really fast for fun ☺! You have seen how 1,469 time-consuming Manual Steps are amazingly replaced with only 2 time-saving Dynamic Automated Steps.

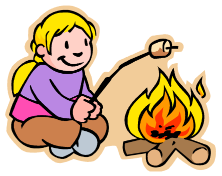
On your future SAS Quests, listen closely to your SAS Intuition and pursue blending your SAS Wishes with the built-in wisdom of SAS. As you experience SAS Wis-h-dom, your research will lead you to your own Happy Accident discoveries which will increase the efficiency of your program designs. As you leave here with your newest BFF in SAS, begin thinking about how you can benefit from this powerful SAS Partnership.

It's not what the world holds for you, it's what YOU bring to it!
Anne of Green Gables

It's not what the SAS World holds for you, it's what YOU bring to it. You are like the language itself ~ you are intuitive and flexible in designing your programs. As a SAS Professional, you are inquisitive, research oriented, and solution driven. Your optimistic and tenacious desire to design a quality program fuels your thoroughness and attention to detail. When you are in your SAS Zone, you are relentless in your pursuit to overcome obstacles and maximize your programming.

Don't be a reservoir, be a river. **John C. Maxwell**

SAS Programming is Mind Art ~ a creative realm where each of you is an Artist. Continue to develop and build on your many skills and talents. Keep looking for different ways to share your God-given abilities and ideas. Don't be a reservoir of SAS knowledge, be a river flowing outward to help and empower other people.

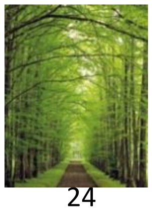


Your life is like a campfire at night – You never know how many people will see it and be comforted and guided by your light.
Claire Draper

Always remember, your contributions make a positive impact in the world. Plan on coming back to the MWSUG Conference next year to shed some light on the exciting things you are learning. All of us are on the SAS journey with you and we look forward to your teaching sessions in the future.

As we conclude, we want to introduce you to our SAS Mascot, Smiley. Smiley represents the SAS Joy which each of us experience as we find better ways to accomplish mighty and worthy deeds using SAS. We hope we have enriched your SAS knowledge. You may not use this amazing SAS Partnership on a daily basis, but when the need arises ~ Oh, how powerful and valuable your relationship will be with your newest BFF in SAS!

Thank You for sharing part of your SAS journey with us ~
☺ Happy SAS Trails to you… until we meet again ☺



MEET THE AUTHORS

Writing is a permanent legacy.

John C. Maxwell

Kent Phelps ~ *Senior Data Governance Analyst, Writer, Teacher, and Coach* **~** has worked in IT and Data Governance since 1990 and has programmed in SAS since 2007. He is a SAS Certified Professional who specializes in blending the best of SAS Enterprise Guide with Base SAS to engineer completely automated solutions, has co-created and led *Intro To SAS EG* classes, offers *SAS News You Can Use*, and has co-authored and presented SAS White Papers at IASUG and MWSUG. He has a B.S. in Electrical Engineering from the University of Nebraska, has studied Transformational Leadership, Dynamic Teamwork, and Personal Growth since 1994, and is certified as a *John Maxwell Team* coach and a *48 Days To The Work You Love* coach. His hope is to encourage and equip you to fulfill your life and leadership potential as you build an enduring legacy of inspiration, excellence, and honor.

****

Ronda Phelps ~ *Writer, Teacher, and Coach* **~** formerly worked in the Banking and Insurance industries for 19 years and has co-authored and presented SAS White Papers at MWSUG. She has studied Transformational Leadership, Dynamic Teamwork, and Personal Growth since 1994, and is certified as a *John Maxwell Team* coach and a *48 Days To The Work You Love* coach. Other past highlights include speaking in Siberia, acting in church productions for over ten years, co-leading and acting in *WOW Drama*, and co-leading a *48 Days To The Work You Love* workshop. She believes YOU are a gift that the world is waiting to receive! Her hope is to encourage and equip you to pursue your unique destiny as you navigate your life journey with intentionality, fulfilling purpose, and enduring hope.

We invite you to share your valued comments with us:

Kent ♥ Ronda Team Phelps
The SASketeers ~ All for SAS & SAS for All!
E-mail: SASketeers@q.com

☺ We look forward to connecting with you in the future ☺

APPENDIX A
Starting Point Information Regarding Syntax For Windows And z/OS

Disclaimer: This presentation details the UNIX syntax for our project example. Please refer to your specific Operating System (e.g. UNIX, Windows, or z/OS) Manual, Installation Configuration, and/or in-house Technical Support for further guidance in how to create the SAS code presented in this paper.

Creating the Dynamic FILENAME Statement on page 8:

```
FILENAME indata '/data/MWSUG/CALL_EXECUTE/file2015*.sas7bdat';
```

- The **Windows** version of the Dynamic **FILENAME** Statement references the specific drive letter along with the path:

```
FILENAME indata "c:\data\MWSUG\CALL_EXECUTE\file2015*.sas7bdat";
```

- The **z/OS** version of the Dynamic **FILENAME** Statement can take different forms depending on the **z/OS** version and installation configuration. Here are **2** reference links as a starting point:
  - **Using the FILENAME Statement or Function to Allocate External Files from SAS® 9.3 Companion for z/OS:**
    http://support.sas.com/documentation/cdl/en/hosto390/65144/HTML/default/viewer.htm#n0yrspsfthx1w5n1gyt6rgzh3qsu.htm
  - **Accessing UNIX System Services Files from SAS® 9.3 Companion for z/OS**:
    http://support.sas.com/documentation/cdl/en/hosto390/65144/HTML/default/viewer.htm#n001udyg5mzcb1n1bhts48m1bal1.htm

Creating the first Dynamic Code append_and_export on page 16:

```
append_and_export = CATS("DATA '/data/MWSUG/CALL_EXECUTE/file_all_",
```

- The **Windows** version of the **append_and_export** Statement uses the specific drive letter:

```
append_and_export = CATS("DATA 'c:\data\MWSUG\CALL_EXECUTE\file_all_",
```

- The **z/OS** version of the **append_and_export** can take different forms depending on the **z/OS** version and installation configuration. Here are **2** reference links as a starting point:
  - **Data Set Options under z/OS from SAS® 9.3 Companion for z/OS**:
    http://support.sas.com/documentation/cdl/en/hosto390/65144/HTML/default/viewer.htm#p1t2wsrhr9x099n1h967cql2j3fm.htm
  - **SAS® 9.3 Companion for z/OS**:
    http://support.sas.com/documentation/cdl/en/hosto390/65144/HTML/default/viewer.htm#titlepage.htm

Executing the first CALL EXECUTE Command on page 21:

```
DATA _NULL_;
    SET path_list_files;
    CALL EXECUTE(fpath_line);
RUN;
```

- The **Windows** version of the **CALL** **EXECUTE** Command is identical in syntax to the **UNIX** version.
- The **z/OS** version of the **CALL** **EXECUTE** Command can take different forms depending on the **z/OS** version and installation configuration even though the **CALL** **EXECUTE** Command is considered to be a portable function in SAS. Here are **2** reference links as a starting point:
    - **CALL EXECUTE Routine from SAS® 9.3 Functions and CALL Routines: Reference**:
      http://support.sas.com/documentation/cdl/en/lefunctionsref/63354/HTML/default/viewer.htm#p1blnvlvciwgs9n0zcilud6d6ei9.htm
    - **SAS® 9.3 Companion for z/OS**:
      http://support.sas.com/documentation/cdl/en/hosto390/65144/HTML/default/viewer.htm#titlepage.htm

APPENDIX B

The Code That Created The Data Sets For Our Project Example

```
DATA '/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat'
     '/data/MWSUG/CALL_EXECUTE/file20150109.sas7bdat'
     '/data/MWSUG/CALL_EXECUTE/file20150116.sas7bdat'
     '/data/MWSUG/CALL_EXECUTE/file20150123.sas7bdat'
     '/data/MWSUG/CALL_EXECUTE/file20150130.sas7bdat'
     '/data/MWSUG/CALL_EXECUTE/file20150206.sas7bdat'
     '/data/MWSUG/CALL_EXECUTE/file20150213.sas7bdat';
   LENGTH Special_Person $20. Special_Number 8. Special_Code $1.;
   INFILE DATALINES DELIMITER=',';
   INPUT Special_Person $ Special_Number Special_Code $;
   SELECT;
      WHEN(_N_ LE 5)  OUTPUT '/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat';
      WHEN(_N_ LE 10) OUTPUT '/data/MWSUG/CALL_EXECUTE/file20150109.sas7bdat';
      WHEN(_N_ LE 15) OUTPUT '/data/MWSUG/CALL_EXECUTE/file20150116.sas7bdat';
      WHEN(_N_ LE 20) OUTPUT '/data/MWSUG/CALL_EXECUTE/file20150123.sas7bdat';
      WHEN(_N_ LE 25) OUTPUT '/data/MWSUG/CALL_EXECUTE/file20150130.sas7bdat';
      WHEN(_N_ LE 30) OUTPUT '/data/MWSUG/CALL_EXECUTE/file20150206.sas7bdat';
      OTHERWISE       OUTPUT '/data/MWSUG/CALL_EXECUTE/file20150213.sas7bdat';
   END;
   DATALINES;
Smiley,10127911,A
Smiley's Son,10173341,K
Smiley's Twin,10376606,B
Smiley's Wife,10927911,A
Smiley's Son,11471884,E
Smiley,10027911,C
Smiley,10877911,H
Smiley's Son,11071884,A
Smiley's Twin,11173691,C
Smiley's Daughter,11375498,J
Smiley,10027911,H
Smiley,10877911,B
Smiley's Son,11071884,F
Smiley's Twin,11173691,H
Smiley's Daughter,11375498,D
Smiley's Son,10173341,G
Smiley,10177911,C
Smiley's Twin,10376606,I
Smiley,10977246,H
Smiley's Son,11471884,A
Smiley's Son,10471884,A
Smiley's Twin,10573616,C
Smiley,10727911,H
Smiley's Son,11571884,F
Smiley's Twin,11773691,H
Smiley,10177911,F
Smiley's Son,10471884,J
Smiley's Twin,10573616,A
Smiley's Son,11571884,D
Smiley's Twin,11773691,F
Smiley,10177911,I
Smiley's Son,10471884,B
Smiley's Twin,10573616,D
Smiley's Son,11571884,G
Smiley's Twin,11773691,I
;
RUN;
```

ACKNOWLEDGMENTS

We want to thank the 26th Annual MWSUG 2015 Tools of the Trade Section Co-Chairs, **Matt Karafa** and **Roger Muller**, for graciously accepting our abstract and paper. In addition, we want to express our appreciation to the Conference Co-Chairs, **Michael G. Wilson** (Academic Chair) and **David Bruckner** (Operations Chair), the Executive Committee and Conference Leaders, and SAS Institute for their diligent efforts in organizing this illuminating and energizing conference.

We also offer our deep gratitude to our friend, mentor, and fellow SASketeer, **Kirk Paul Lafler**. Your heart to continuously share what you are learning, blended with your servant leadership and supportive guidance, is a constant light of encouragement to us. You inspire us to share what we are learning and our hope is to be a light of encouragement to you as well **~ All for SAS & SAS for All.**

REFERENCES

Agarwal, Megha (2012), *The Power of "The FILENAME" Statement*, Gilead Sciences, Foster City, CA, USA.
http://www.lexjansen.com/wuss/2012/63.pdf

Gan, Lu (2012), *Using SAS® to Locate and Rename External Files*, Pharmaceutical Product Development, L.L.C., Austin, TX, USA.
http://www.scsug.org/wp-content/uploads/2012/11/Using-SAS-to-locate-and-rename-external-files.pdf

Hamilton, Jack (2012), *Obtaining a List of Files in a Directory Using SAS® Functions*. http://www.wuss.org/proceedings12/55.pdf

Lafler, Kirk Paul and Charles Edwin Shipp (2012), *Google® Search Tips and Techniques for SAS® and JMP® Users*, Proceedings of the 23rd Annual MidWest SAS Users Group (MWSUG) 2012 Conference, Software Intelligence Corporation, Spring Valley, CA, and Consider Consulting, Inc., San Pedro, CA, USA.
http://www.mwsug.org/proceedings/2012/JM/MWSUG-2012-JM06.pdf

Langston, Rick (2013), *Submitting SAS® Code On The Side*; SAS Institute Inc., Cary, NC.
http://support.sas.com/resources/papers/proceedings13/032-2013.pdf

Michel, Denis (2005), *CALL EXECUTE: A Powerful Data Management Tool*, Proceedings of the 30th Annual SAS® Users Group International (SUGI) 2005 Conference, Johnson & Johnson Pharmaceutical Research and Development, L.L.C.
http://www2.sas.com/proceedings/sugi30/027-30.pdf

Phelps, Kent ♥ Ronda Team (2015), *The Joinless Join ~ The Impossible Dream Come True; Expanding the Power of SAS® Enterprise Guide® in a New Way*, Proceedings of the 26th Annual MidWest SAS Users Group (MWSUG) 2015 Conference, The SASketeers, Des Moines, IA, USA.

Phelps, Kent ♥ Ronda Team and Kirk Paul Lafler (2014), *SAS® Commands PIPE and CALL EXECUTE; Dynamically Advancing From Strangers to Your Newest BFF (Best Friends Forever)*, Proceedings of the 25th Annual MidWest SAS Users Group (MWSUG) 2014 Conference, The SASketeers, Des Moines, IA, and Software Intelligence Corporation, Spring Valley, CA, USA.
http://www.mwsug.org/proceedings/2014/BI/MWSUG-2014-BI13.pdf

Phelps, Kent ♥ Ronda Team and Kirk Paul Lafler (2014), *The Joinless Join; Expand the Power of SAS® Enterprise Guide® in a New Way*, Proceedings of the 25th Annual MidWest SAS Users Group (MWSUG) 2014 Conference, The SASketeers, Des Moines, IA, and Software Intelligence Corporation, Spring Valley, CA, USA.
http://www.mwsug.org/proceedings/2014/BI/MWSUG-2014-BI12.pdf

Phelps, Kent ♥ Ronda Team and Kirk Paul Lafler (2013), *SAS® Commands PIPE and CALL EXECUTE; Dynamically Advancing From Strangers to Best Friends*, Presented at Iowa SAS Users Group (IASUG), The SASketeers, Des Moines, IA, and Software Intelligence Corporation, Spring Valley, CA, USA.

Phelps, Kent ♥ Ronda Team and Kirk Paul Lafler (2013), *The Joinless Join; Expand the Power of SAS® Enterprise Guide® in a New Way*, Presented at Iowa SAS Users Group (IASUG), The SASketeers, Des Moines, IA, and Software Intelligence Corporation, Spring Valley, CA, USA.

Phelps, Kent ♥ Ronda Team and Kirk Paul Lafler (2013), *SAS® Commands PIPE and CALL EXECUTE; Dynamically Advancing From Strangers to Best Friends*, Proceedings of the 24th Annual MidWest SAS Users Group (MWSUG) 2013 Conference, The SASketeers, Des Moines, IA, and Software Intelligence Corporation, Spring Valley, CA, USA.
http://www.mwsug.org/proceedings/2013/00/MWSUG-2013-0003.pdf

Phelps, Kent ♥ Ronda Team and Kirk Paul Lafler (2013), *The Joinless Join; Expand the Power of SAS® Enterprise Guide® in a New Way*, Proceedings of the 24th Annual MidWest SAS Users Group (MWSUG) 2013 Conference, The SASketeers, Des Moines, IA, and Software Intelligence Corporation, Spring Valley, CA, USA. http://www.mwsug.org/proceedings/2013/BB/MWSUG-2013-BB06.pdf

SAS Institute Inc. (2012), *SAS® 9.3 Companion for z/OS, Second Edition*; Cary, NC; SAS Institute Inc.
http://support.sas.com/documentation/cdl/en/hosto390/65144/HTML/default/viewer.htm#titlepage.htm

SAS Institute Inc. (2011), *SAS® 9.3 Functions and CALL Routines: Reference*; Cary, NC; SAS Institute Inc.
http://support.sas.com/documentation/cdl/en/lefunctionsref/63354/HTML/default/viewer.htm#titlepage.htm

Spector, Phil, *An Introduction to the SAS System*; Statistical Computing Facility; University of California, Berkeley.
http://www.stat.berkeley.edu/~spector/

Support.SAS.com (2007), *Using FILEVAR= to Read Multiple External Files in a DATA Step*.
http://support.sas.com/techsup/technote/ts581.pdf

Varney, Brian (2008), *You Check out These Pipes: Using Microsoft Windows Commands from SAS®*, SAS Institute Inc. 2008. Proceedings of the SAS® Global Forum 2008 Conference, Cary, NC; SAS Institute Inc.
http://www2.sas.com/proceedings/forum2008/092-2008.pdf

Watson, Richann (2013), *Let SAS® Do Your DIRty Work*, Experis, Batavia, OH.
http://www.pharmasug.org/proceedings/2013/TF/PharmaSUG-2013-TF06.pdf

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. The symbol, ®, indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

DISCLAIMER

We have endeavored to provide accurate and helpful information in this SAS White Paper. The information is provided in 'Good Faith' and 'As Is' without any kind of warranty, either expressed or implied. Recipients acknowledge and agree that we and/or our companies are not, and never will be, liable for any problems and/or damages whatsoever which may arise from the recipient's use of the information in this paper. Please refer to your specific Operating System (e.g. UNIX, Windows, or z/OS) Manual, Installation Configuration, and/or in-house Technical Support for further guidance in how to create the SAS code presented in this paper.