

Base SAS® and SAS® Enterprise Guide® ~ Automate Your SAS World with Dynamic Code; *Your Newest BFF (Best Friend 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 SAS relationship with the Server, PC, or Mainframe. To communicate more efficiently ~ and to increasingly automate your SAS World ~ you will want to learn how to transform Static Code into Dynamic Code that automatically recreates the Static Code, and then executes the recreated Static Code automatically.

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

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 Friend Forever) in SAS. Please see the Appendices to review starting point information regarding the syntax for Windows and z/OS, and to review the source 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 all of you know, SAS enables you to creatively program **Smarter And Smarter**. However, SAS, as remarkable as it is, will remain an island unto itself without your coding proficiency.

The tagline for SAS is *The Power To Know*® and your 'power to know' greatly expands with your determination to communicate more efficiently 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** ~ but these powers will quickly go down the drain if you do not continuously learn how to request data more efficiently and how to automate your SAS World.

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

- ❖ How do I request data more efficiently 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 a Base SAS Program Node which:

- ❖ Transforms a Static FILENAME Statement into a Dynamic FILENAME Statement to obtain a Directory Listing of a date range of files from the server.
- ❖ Utilizes 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.
 - The Dynamic Code in this presentation recreates Static Code which is executable code that never changes and always runs exactly the same way.
 - The Dynamic Code will store the recreated Static Code in a variable in a SAS dataset.
- ❖ Executes the recreated 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 using the INDSNAME SET Option

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
Dynamic Code
can become your Best Friend Forever in SAS.**

😊 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 Dynamic Code with a Dynamic FILENAME Statement, the INDSNAME SET Option, 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, when we started the following project with one plan in mind, we soon found that in order to overcome obstacle bumps on the project road, we needed 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.
- ❖ **Append** the **52** weekly snapshot 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.

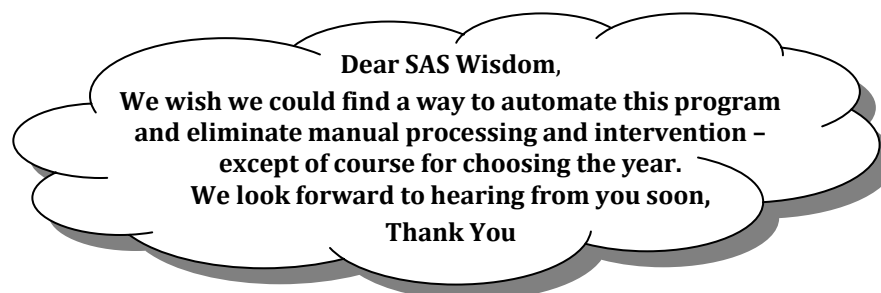
Since SAS Enterprise Guide was being used to design this project, the first decision to make was, “Should the program be designed with Graphical User Interface (GUI) and/or a Base SAS Program Node?”

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

- ❖ What will it take to **manually** add **52** weekly snapshot 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!”



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 **1** DATA step to read and append **52** data sets together?
- ❖ What will it take to **automatically** extract vital variables in **1** DATA step?
- ❖ What will it take to **automatically** enter the derived value of the Load_Date variable in **1** DATA step?
- ❖ 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 **1** DATA step – **12** more times – while running the program for the **13** year timeframe?

We began a quest to accomplish the grand and noble deed of automating this program ☺. Our first task was to find a way to transform a Static FILENAME Statement into a Dynamic FILENAME Statement to read **52** weekly snapshot 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 many different ways to transform a Static FILENAME Statement into a Dynamic FILENAME Statement to automatically and sequentially read the content of multiple data sets.

Obstacle Bump ~ Unfortunately this article did not mention how to use a Dynamic FILENAME Statement to obtain a Directory Listing of the filename of each data set while reading multiple data sets ~ Bummer!

☺ **Happy Accident Alert** ☺ ~ We did not give up and began a series of researching detours. Along the way we finally discovered that when a Dynamic FILENAME Statement is used, SAS will actually assign a variable called FILENAME to the name of each file being read ~ Yea!

This knowledge enabled us to transform a Static FILENAME Statement into a Dynamic FILENAME Statement to obtain a Directory Listing of the filenames which can 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.

Obstacle Bump ~ However, we then realized that although a Dynamic FILENAME Statement can be used to obtain a Directory Listing which is utilized to transform Static Code into Dynamic Code that automatically recreates the Static Code, we determined that the same Dynamic FILENAME Statement could not be used again within the recreated Static Code to obtain the name of each data set as it is actually being read.

Our SAS Intuition pondered once again, “Surely, when multiple input data sets are used as inputs in a DATA step, there must be a way to obtain the name of the data set from where each input observation is read!”

😊 **Happy Accident Alert** 😊 ~ Another Google search happily uncovered a SET option called INDSNAME which identifies the input data set being read with each input observation.

We concluded that a variable called FILENAME can be used to identify the name of an input data set when using a Dynamic FILENAME Statement, and a variable called INDSNAME can be used to identify the name of the input data set when using a Static input SET Statement using the INDSNAME SET option. We will demonstrate the differences as we review the project example.

Learning this information enabled us to design a program which utilizes:

- ❖ A Dynamic FILENAME Statement to obtain one Directory Listing per year for **13** years of the filenames of the **52** weekly snapshot data sets.
- ❖ The Directory Listing to transform **Extract, Append, and Export 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 using the INDSNAME SET Option.
 - Append the **52** weekly snapshot 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.

Once the program has run, the recreated Extract, Append, and Export Static Code can be run **manually** by copying and pasting the code into another Program Node. This program fulfills most of the project requirements... but remember, our SAS Wish was to **COMPLETELY** automate this project.



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 recreated Static Code automatically?" You guessed it... our SAS Intuition spoke up 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 hopeful Google search led to a White Paper titled *CALL EXECUTE: A Powerful Data Management Tool* which revealed that a CALL EXECUTE command already existed!

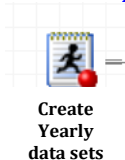
Here is a brief overview of the White Paper:

- ❖ 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!



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 program will run automatically without any manual processing or intervention ~ except for choosing the year!

Here is the program displayed as a SAS Enterprise Guide Base SAS Program Node:



As you can see from this SAS Quest, it pays to listen to your SAS Intuition. A series of 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 there is a treasure chest 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

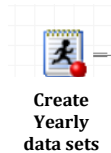
THE POWER TO KNOW

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.

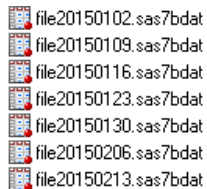
This code will obtain and store the Directory Listing:



```
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 this Directory Listing is used to transform Static Code into Dynamic Code.

THE POWER TO TRANSFORM

Static Code Into Dynamic Code Using The INDSNAME SET Option

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

This is the original Extract Static Code for weeks 1 and 7:

```
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;
```

```
DATA file_final_20150213;  
  SET '/data/MWSUG/CALL_EXECUTE/file20150213.sas7bdat';  
  FORMAT Load_Date date9.; Load_Date = '13FEB2015'd;  
  KEEP Special_Person Special_Number Load_Date;  
RUN;
```

- ❖ Each weekly **DATA** step creates a `file_final_YYYYMMDD` data set with the `YYYYMMDD` matching the create date of the data set.
- ❖ The `Load_Date` variable is derived and formatted as a SAS date (`date9`) which also matches the date of the data set.
- ❖ A **KEEP** statement is used to keep the `Special_Person` and `Special_Number` from the data set while also keeping the `Load_Date` which is derived within the **DATA** step.

Here is the Append Code combined with the Export Code for weeks 1 to 7:

```
DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';  
  SET file_final_20150102.sas7bdat  
    file_final_20150109.sas7bdat  
    file_final_20150116.sas7bdat  
    file_final_20150123.sas7bdat  
    file_final_20150130.sas7bdat  
    file_final_20150206.sas7bdat  
    file_final_20150213.sas7bdat;  
RUN;
```

- ❖ Each yearly **DATA** step creates a `file_final_YYYY` data set with the `YYYY` matching the year of each data set.
- ❖ Each of the `file_final_YYYYMMDD` data sets are **SET** as data sets one after another.
- ❖ Question: How can the Extract, Append, and Export code be completely combined into one **DATA** step?

Here is the Extract, Append, and Export Code almost completely combined:

```
DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
  SET '/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';
  FORMAT Load_Date date9.; Load_Date = '02JAN2015'd;
  KEEP Special_Person Special_Number Load_Date;
RUN;
```

- ❖ The major change from the previous Append is that this Append **SETS** the 52 original data sets as inputs rather than using the outputs from the 52 Extract **DATA** steps.
- ❖ Question: Since the Dynamic FILENAME Statement, which is used to execute the Dynamic Code that recreates this Static Code, is not available during the runtime of this Static Code, how do we obtain the Load_Date?

Here is the Extract, Append, and Export Code completely combined:

```
DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
  LENGTH Current_SAS_dataset $100;
  SET '/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'
      INDSNAME=Current_SAS_dataset;
  FORMAT Load_Date date9.;
  KEEP Special_Person Special_Number Load_Date;
  Load_Date = MDY(INPUT(SUBSTR(Current_SAS_dataset,34,2),2.),
                  INPUT(SUBSTR(Current_SAS_dataset,36,2),2.),
                  INPUT(SUBSTR(Current_SAS_dataset,30,4),4.));
RUN;
```

- ❖ Create a variable `Current_SAS_dataset` with a `LENGTH` long enough for each data set name and path in the Directory Listing.
- ❖ Place the `LENGTH` before the `SET` statement so the complete data set name and path are captured.
- ❖ Add `INDSNAME=Current_SAS_dataset` to the end of the `SET` statement so `Current_SAS_dataset` will always be assigned the data set name and path of the observation being read.
- ❖ Use the `MDY`, `INPUT`, and `SUBSTR` functions to transform the month, day, and year of each data set name and path (`Current_SAS_dataset`) into the `Load_Date`.

Here is the Extract, Append, and Export Code efficiently combined:

```
DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
  LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
  SET '/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'
      INDSNAME=Current_SAS_dataset;
  FORMAT Load_Date date9.;
  KEEP Special_Person Special_Number Load_Date;
  RETAIN Load_Date;
  IF Current_SAS_dataset NE Prior_SAS_dataset THEN
    DO;
      Load_Date = MDY(INPUT(SUBSTR(Current_SAS_dataset,34,2),2.),
                     INPUT(SUBSTR(Current_SAS_dataset,36,2),2.),
                     INPUT(SUBSTR(Current_SAS_dataset,30,4),4.));
      Prior_SAS_dataset = Current_SAS_dataset;
    END;
RUN;
```

- ❖ The previous **DATA** step correctly derives and assigns the **Load_Date** with each observation.
- ❖ However, for better efficiency, we can create another variable **Prior_SAS_dataset** to track the data set changes and add a **RETAIN Load_Date** statement so that we only derive **Load_Date** with each observation when the data set changes (**IF Current_SAS_dataset NE Prior_SAS_dataset**).
- ❖ The next step is to transform the efficient Static Code into Dynamic Code.

How to transform the efficient 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 '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
          LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
          SET '/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'
            INDSNAME=Current_SAS_dataset;
          FORMAT Load_Date date9.;
          KEEP Special_Person Special_Number Load_Date;
          RETAIN Load_Date;
          IF Current_SAS_dataset NE Prior_SAS_dataset THEN
            DO;
              Load_Date = MDY(INPUT(SUBSTR(Current_SAS_dataset,34,2),2.),
                              INPUT(SUBSTR(Current_SAS_dataset,36,2),2.),
                              INPUT(SUBSTR(Current_SAS_dataset,30,4),4.));
              Prior_SAS_dataset = Current_SAS_dataset;
            END;
          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.

Next identify what changes with each observation of 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;
fpath_line = CATS("
  DATA '/data/MWSUG/CALL_EXECUTE/file_final ", "2015", ".sas7bdat';
  LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
  SET '"/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", "'
      INDSNAME=Current_SAS_dataset;
  FORMAT Load_Date date9.;
  KEEP Special_Person Special_Number Load_Date;
  RETAIN Load_Date;
  IF Current_SAS_dataset NE Prior_SAS_dataset THEN
    DO;
      Load_Date = MDY(INPUT(SUBSTR(Current_SAS_dataset,34,2),2.),
                      INPUT(SUBSTR(Current_SAS_dataset,36,2),2.),
                      INPUT(SUBSTR(Current_SAS_dataset,30,4),4.));
      Prior_SAS_dataset = Current_SAS_dataset;
    END;
  RUN;");

  OUTPUT;
END;
RUN;
```

- ❖ Surround the year of the output data set with double quotes and commas because the year of the output data set will match the year of the input data sets.
- ❖ Surround the names of the input data sets with double quotes and commas because the names of the input data sets will change with each observation.

Now identify the timing of changes with each observation of 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;
fpath_line = CATS("
  DATA '/data/MWSUG/CALL_EXECUTE/file_final_", "2015", ".sas7bdat';
  LENGTH Current SAS_dataset Prior SAS_dataset $100;
  SET '"', "/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", "'
  INDSNAME=Current SAS_dataset;
  FORMAT Load_Date date9.;
  KEEP Special_Person Special_Number Load_Date;
  RETAIN Load_Date;
  IF Current SAS_dataset NE Prior SAS_dataset THEN
    DO;
      Load_Date = MDY(INPUT(SUBSTR(Current_SAS_dataset,34,2),2.),
                      INPUT(SUBSTR(Current_SAS_dataset,36,2),2.),
                      INPUT(SUBSTR(Current_SAS_dataset,30,4),4.));
      Prior_SAS_dataset = Current_SAS_dataset;
    END;
  RUN;");
  OUTPUT;
END;
RUN;

```

- ❖ The `DATA` through `SET` statements needs to occur only once, and `2015` can be derived from the first observation of `fpath` which is derived from `SAS_data_set_and_path`.
- ❖ The `/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat` through `file20150213.sas7bdat` input data sets needs to occur with each observation because they are equivalent to each observation of `fpath`.
- ❖ The `INDSNAME=Current SAS_dataset;` through `RUN;` statements needs to occur at the end after all the observations of the Directory Listing have been read and processed as the `fpath` variable.

Code for the timing of changes with each observation of 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 fpath_line;
  FORMAT Load_Date date9.;
  INFILE indata TRUNCOVER FILENAME=SAS_data_set_and_path END=DONE;
  INPUT;
  IF _N_ = 1 THEN
    fpath_line =
      CATS("DATA '/data/MWSUG/CALL_EXECUTE/file_final_",
          SUBSTR(SAS_data_set_and_path,30,4),".sas7bdat'",
          LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
          SET ";
  IF fpath NE SAS_data_set_and_path THEN
    DO;
      fpath = SAS_data_set_and_path;
      fpath_line = CATS(fpath_line)||" "||CATS(fpath)||" ";
    END;
  IF DONE THEN
    DO;
      fpath_line = CATS(fpath_line)||
        "INDSNAME=Current_SAS_dataset;
        FORMAT Load_Date date9.;
        KEEP Special_Person Special_Number Load_Date;
        RETAIN Load_Date;
        IF Current_SAS_dataset NE Prior_SAS_dataset THEN
          DO;
            Load_Date = MDY(INPUT(SUBSTR(Current_SAS_dataset,34,2),2.),
                            INPUT(SUBSTR(Current_SAS_dataset,36,2),2.),
                            INPUT(SUBSTR(Current_SAS_dataset,30,4),4.));
            Prior_SAS_dataset = Current_SAS_dataset;
          END;
        RUN;" );
      OUTPUT;
    END;
  RUN;

```

- ❖ Add the `fpath_line` variable to the `RETAIN` statement to enable concatenating each piece of the `DATA` step to the `fpath_line` variable until the result is the original Static Code.
- ❖ Begin what must occur with the first observation with `IF _N_ = 1 THEN`, and change `2015` to the derivation of `SUBSTR(SAS_data_set_and_path,30,4)`; noting that `SAS_data_set_and_path` is assigned the name of each input data set and path by the `FILENAME=SAS_data_set_and_path` option on the `INFILE` statement.
- ❖ Next we add the `IF fpath NE SAS_data_set_and_path THEN` logic so that only when the first observation of each new input data set is read will the `fpath = SAS_data_set_and_path` assignment occur.
- ❖ Assign `fpath_line` to itself with beginning and ending spaces removed (`CATS(fpath_line)`) concatenated (`||`) to each input data set and path (`CATS(fpath)`) surrounded by single quotes (`" "`).
- ❖ Add `END=DONE` to the `INFILE` Statement so `DONE` will be set to True once the last observation is read from the last input data set.
- ❖ Assign `fpath_line` to itself with beginning and ending spaces removed (`CATS(fpath_line)`) concatenated (`||`) to the remainder of the code needed to complete the `DATA` step.
- ❖ Add the `OUTPUT` Statement to create only one observation of the `fpath_line` variable containing the entire resolved and concatenated original Static Code.

This is the final Dynamic Code which recreates the original 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 fpath_line;
  FORMAT Load_Date date9.;
  INFILE indata TRUNCOVER FILENAME=SAS_data_set_and_path END=DONE;
  INPUT;
  IF _N_ = 1 THEN
    fpath_line =
      CATS("DATA '/data/MWSUG/CALL_EXECUTE/file_final_",
          SUBSTR(SAS_data_set_and_path,30,4), ".sas7bdat';
          LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
          SET ");
  IF fpath NE SAS_data_set_and_path THEN
    DO;
      fpath = SAS_data_set_and_path;
      fpath_line = CATS(fpath_line)||" '"||CATS(fpath)||"' ";
    END;
  IF DONE THEN
    DO;
      fpath_line = CATS(fpath_line)||
        "INDSNAME=Current_SAS_dataset;
        FORMAT Load_Date date9.;
        KEEP Special_Person Special_Number Load_Date;
        RETAIN Load_Date;
        IF Current_SAS_dataset NE Prior_SAS_dataset THEN
          DO;
            Load_Date = MDY(INPUT(SUBSTR(Current_SAS_dataset,34,2),2.),
                            INPUT(SUBSTR(Current_SAS_dataset,36,2),2.),
                            INPUT(SUBSTR(Current_SAS_dataset,30,4),4.));
            Prior_SAS_dataset = Current_SAS_dataset;
          END;
        RUN;");
      OUTPUT;
    END;
  RUN;
```

- ❖ Now that the Dynamic Code has been finalized, we will do a recap of our exciting journey and then we will demonstrate how the resulting Static Code is executed using the CALL EXECUTE command.

This is the program displayed as a Base SAS Program Node:



The Final Dynamic Code and the resulting Final Recreated Static Code:

Dynamic Code within the Base SAS Program Node:

```
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 fpath_line;  
FORMAT Load_Date date9.;  
INFILE indata TRUNCOVER FILENAME=  
SAS_data_set_and_path END=DONE;  
INPUT;  
IF _N_ = 1 THEN fpath_line =  
CATS("DATA '/data/MWSUG/CALL_EXECUTE/  
file_final_",  
SUBSTR(SAS_data_set_and_path,30,4),  
".sas7bdat";  
LENGTH Current_SAS_dataset  
Prior_SAS_dataset $100;  
SET ";  
IF fpath NE SAS_data_set_and_path THEN  
DO;  
fpath = SAS_data_set_and_path;  
fpath_line = CATS(fpath_line)||"  
"||CATS(fpath)||" ";  
END;  
IF DONE THEN  
DO;  
fpath_line = CATS(fpath_line)||  
"INDSNAME=Current_SAS_dataset;  
FORMAT Load_Date date9.;  
KEEP Special_Person Special_Number Load_Date;  
RETAIN Load_Date;  
IF Current_SAS_dataset NE Prior_SAS_dataset  
THEN  
DO;  
Load_Date =  
MDY(INPUT(SUBSTR(Current_SAS_dataset,34,2),2.),  
INPUT(SUBSTR(Current_SAS_dataset,36,2),2.),  
INPUT(SUBSTR(Current_SAS_dataset,30,4),4.));  
Prior_SAS_dataset = Current_SAS_dataset;  
END;  
RUN;");  
OUTPUT;  
END;  
RUN;
```

Static Code recreated by running the Dynamic Code:

```
DATA '/data/MWSUG/CALL_EXECUTE/  
file_final_2015.sas7bdat';  
LENGTH Current_SAS_dataset  
Prior_SAS_dataset $100;  
SET  
'/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'  
INDSNAME=Current_SAS_dataset;  
FORMAT Load_Date date9.;  
KEEP Special_Person Special_Number Load_Date;  
RETAIN Load_Date;  
IF Current_SAS_dataset NE Prior_SAS_dataset THEN  
DO;  
Load_Date =  
MDY(INPUT(SUBSTR(Current_SAS_dataset,34,2),2.),  
INPUT(SUBSTR(Current_SAS_dataset,36,2),2.),  
INPUT(SUBSTR(Current_SAS_dataset,30,4),4.));  
Prior_SAS_dataset = Current_SAS_dataset;  
END;  
RUN;
```

THE POWER TO TRANSFORM section has walked us through the process of transforming **Extract, Append, and Export Static Code** into Dynamic Code that automatically recreates the Static Code to Extract vital variables from **52** weekly snapshot data sets and combining them with a Load_Date variable (created from the Friday date value derived from the filenames of the data sets), appending the **52** weekly snapshot data sets together to create a yearly data set, and exporting the yearly data set back to the server.

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 yearly data set, the CALL EXECUTE Command is used to execute the recreated Static Code automatically.

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

```

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

- ❖ The **DATA** step 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.
- ❖ The **CALL EXECUTE** Command runs the **fpath_line** variable in the **path_list_files** data set.

Here is the code with the value of the fpath_line variable resolved:

```

DATA _NULL_;
  SET path_list_files;
  CALL EXECUTE(
    DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
    LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
    SET '/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'
    INDSNAME=Current_SAS_dataset;
    FORMAT Load_Date date9.;
    KEEP Special_Person Special_Number Load_Date;
    RETAIN Load_Date;
    IF Current_SAS_dataset NE Prior_SAS_dataset THEN
      DO;
        Load_Date = MDY(INPUT(SUBSTR(Current_SAS_dataset,34,2),2.),
                        INPUT(SUBSTR(Current_SAS_dataset,36,2),2.),
                        INPUT(SUBSTR(Current_SAS_dataset,30,4),4.));
        Prior_SAS_dataset = Current_SAS_dataset;
      END;
    RUN;
  );
RUN;
```

- ❖ Next we will demonstrate how this code is resolved as the data sets change in the **SET** statement.

Here is the processing of the 1st data set of the SET statement:

```
DATA _NULL_ ;
  SET path_list_files;
  CALL EXECUTE(
    DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
    LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
    SET '/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'
    INDSNAME=Current_SAS_dataset;
    FORMAT Load_Date date9 ;
    KEEP Special_Person Special_Number Load_Date;
    RETAIN Load_Date;
    IF Current_SAS_dataset NE Prior_SAS_dataset THEN
      DO;
        Load_Date = MDY( INPUT( SUBSTR( Current_SAS_dataset, 34, 2), 2.),
                        INPUT( SUBSTR( Current_SAS_dataset, 36, 2), 2.),
                        INPUT( SUBSTR( Current_SAS_dataset, 30, 4), 4.));
        Prior_SAS_dataset = Current_SAS_dataset;
      END;
    RUN;
  );
RUN;
```

- ❖ The `INDSNAME=Current_SAS_dataset` `SET` option assigns `Current_SAS_dataset` to equal the first data set and path which is `SET '/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat'`.
- ❖ The `IF Current_SAS_dataset NE Prior_SAS_dataset THEN` logic is only True when the first observation of each data set is read because of the `Prior_SAS_dataset = Current_SAS_dataset` assignment.
- ❖ The `Load_Date` is assigned by the `SUBSTR` function capturing the character month, day, and year from the `Current_SAS_dataset`.
- ❖ The character month, day, and year are converted to numeric using the `INPUT` function and then converted to a SAS date using the `MDY` function and formatted as a SAS date by the `FORMAT Load_Date date9.`
- ❖ The `Load_Date` is retained by the `RETAIN Load_Date` statement and reassigned with the first observation of each new input data set.

Here is the beginning of resolving the SUBSTR function of the Load_Date:

```
DATA _NULL_ ;
  SET path_list_files;
  CALL EXECUTE (
    DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
    LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
    SET '/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'
      INDSNAME=Current_SAS_dataset;
    FORMAT Load_Date date9.;
    KEEP Special_Person Special_Number Load_Date;
    RETAIN Load_Date;
    IF Current_SAS_dataset NE Prior_SAS_dataset THEN
      DO;
        Load_Date = MDY(INPUT(SUBSTR(Current_SAS_dataset,34,2),2.),
                        INPUT(SUBSTR(Current_SAS_dataset,36,2),2.),
                        INPUT(SUBSTR(Current_SAS_dataset,30,4),4.));
        Prior_SAS_dataset = Current_SAS_dataset;
      END;
    RUN;
  );
RUN;
```

- ❖ The `Load_Date` resolution begins by resolving the `SUBSTR`'s of month, day, and year from the `Current_SAS_dataset` variable.
- ❖ The `SUBSTR(Current_SAS_dataset,34,2)` resolves to `'01'` for the month as a character.
- ❖ The `SUBSTR(Current_SAS_dataset,36,2)` resolves to `'02'` for the day as a character.
- ❖ The `SUBSTR(Current_SAS_dataset,30,4)` resolves to `'2015'` for the year as a character.

Here is the resolution of the SUBSTR function of month, day, and year:

```
DATA _NULL_ ;
  SET path_list_files;
  CALL EXECUTE (
    DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
    LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
    SET '/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'
        INDSNAME=Current_SAS_dataset;
    FORMAT Load_Date date9.;
    KEEP Special_Person Special_Number Load_Date;
    RETAIN Load_Date;
    IF Current_SAS_dataset NE Prior_SAS_dataset THEN
      DO;
        Load_Date = MDY(INPUT('01',2.),
                        INPUT('02',2.),
                        INPUT('2015',4.));
        Prior_SAS_dataset = Current_SAS_dataset;
      END;
    RUN;
  );
RUN;
```

- ❖ The SUBSTR function captured the character month, day, and year from the Current_SAS_dataset.

Here is the resolution of the INPUT function of month, day, and year:

```
DATA _NULL_ ;
  SET path_list_files;
  CALL EXECUTE (
    DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
    LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
    SET '/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'
        INDSNAME=Current_SAS_dataset;
    FORMAT Load_Date date9.;
    KEEP Special_Person Special_Number Load_Date;
    RETAIN Load_Date;
    IF Current_SAS_dataset NE Prior_SAS_dataset THEN
      DO;
        Load_Date = MDY(01,02,2015);
        Prior_SAS_dataset = Current_SAS_dataset;
      END;
    RUN;
  );
RUN;
```

- ❖ The INPUT function converted the character month, day, and year to numeric.

Here is the final resolution of the Load_Date for the 1st input data set:

```
DATA _NULL_ ;
  SET path_list_files;
  CALL EXECUTE (
    DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
    LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
    SET '/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'
      INDSNAME=Current_SAS_dataset;
    FORMAT Load_Date date9.;
    KEEP Special_Person Special_Number Load_Date;
    RETAIN Load_Date;
    IF Current_SAS_dataset NE Prior_SAS_dataset THEN
      DO;
        Load_Date = '02JAN2015'd;
        Prior_SAS_dataset = Current_SAS_dataset;
      END;
  RUN;
);
RUN;
```

- ❖ The MDY function and `FORMAT Load_Date date9.` converts the month, day, and year to a SAS date.

Here is the final resolution of the Load_Date for the 2nd input data set:

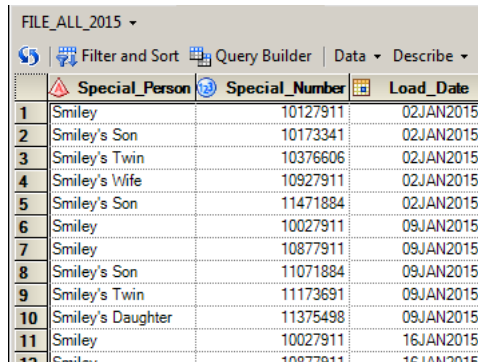
```
DATA _NULL_ ;
  SET path_list_files;
  CALL EXECUTE (
    DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
    LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
    SET '/data/MWSUG/CALL_EXECUTE/file20150102.sas7bdat'
      '/data/MWSUG/CALL_EXECUTE/file20150109.sas7bdat'
      '/data/MWSUG/CALL_EXECUTE/file20150116.sas7bdat'
      '/data/MWSUG/CALL_EXECUTE/file20150123.sas7bdat'
      '/data/MWSUG/CALL_EXECUTE/file20150130.sas7bdat'
      '/data/MWSUG/CALL_EXECUTE/file20150206.sas7bdat'
      '/data/MWSUG/CALL_EXECUTE/file20150213.sas7bdat'
      INDSNAME=Current_SAS_dataset;
    FORMAT Load_Date date9.;
    KEEP Special_Person Special_Number Load_Date;
    RETAIN Load_Date;
    IF Current_SAS_dataset NE Prior_SAS_dataset THEN
      DO;
        Load_Date = '09JAN2015'd;
        Prior_SAS_dataset = Current_SAS_dataset;
      END;
  RUN;
);
RUN;
```

- ❖ The MDY function and `FORMAT Load_Date date9.` converts the month, day, and year to a SAS date.

Here is the final resolution of the Load_Date for the last input data set:

```
DATA _NULL_ ;
  SET path_list_files;
  CALL EXECUTE(
    DATA '/data/MWSUG/CALL_EXECUTE/file_final_2015.sas7bdat';
    LENGTH Current_SAS_dataset Prior_SAS_dataset $100;
    SET '/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'
    INDSNAME=Current_SAS_dataset;
    FORMAT Load_Date date9.;
    KEEP Special_Person Special_Number Load_Date;
    RETAIN Load_Date;
    IF Current_SAS_dataset NE Prior_SAS_dataset THEN
      DO;
        Load Date = '02FEB2015'd;
        Prior_SAS_dataset = Current_SAS_dataset;
      END;
  RUN;
);
RUN;
```

- ❖ The MDY function and `FORMAT Load_Date date9.` converts the month, day, and year to a SAS date.
- ❖ Here is the result of executing `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
3	Smiley's Twin	10376606	02JAN2015
4	Smiley's Wife	10927911	02JAN2015
5	Smiley's Son	11471884	02JAN2015
6	Smiley	10027911	09JAN2015
7	Smiley	10877911	09JAN2015
8	Smiley's Son	11071884	09JAN2015
9	Smiley's Twin	11173691	09JAN2015
10	Smiley's Daughter	11375498	09JAN2015
11	Smiley	10027911	16JAN2015

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 Create Yearly data set program node, rerun the program node, 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 Create Yearly data set program node and rerun 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 using the INDSNAME SET Option 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 **1** time-saving Dynamic Automated Step.

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 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 powerful 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 ~ SAS Certified Professional ~ B.S. Electrical Engineering ~ Writer ~ Teacher ~ Coach ~ has presented at the MWSUG Conference for 3 years, worked in IT and Data Governance since 1990, programmed in SAS since 2007, and specializes in blending the best of Base SAS with SAS Enterprise Guide to engineer automated solutions. He co-created/taught *Intro to SAS EG* classes, offered *SAS News You Can Use*, presented at the Iowa SAS Users Group (IASUG), studied Transformational Leadership, Dynamic Teamwork, and Personal Growth since 1994, and is certified as a *John Maxwell Team* and *48 Days To The Work You Love* Coach. Past highlights include acting for over ten years, co-leading *WOW Drama*, singing a drama solo with a live orchestra, and auditioning in Branson, MO. Kent wants 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 ~ Coach ~ has presented at the MWSUG Conference for 2 years, formerly worked in the Banking and Insurance industries for 19 years, studied Transformational Leadership, Dynamic Teamwork, and Personal Growth since 1994, and is certified as a *John Maxwell Team* and *48 Days To The Work You Love* Coach. Past highlights include speaking in Siberia, acting for over ten years, co-leading *WOW Drama*, and developing life-changing presentations. Ronda believes that YOU are a gift the world is waiting to receive, and she wants 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® 9.4 Companion for z/OS:**

<http://support.sas.com/documentation/cdl/en/hosto390/68955/HTML/default/viewer.htm#n0yrspsfthx1w5n1gyt6rgzh3qsu.htm>

- **Accessing UNIX System Services Files from SAS® 9.4 Companion for z/OS:**

<http://support.sas.com/documentation/cdl/en/hosto390/68955/HTML/default/viewer.htm#n001udyg5mzcb1n1bhbs48m1bal1.htm>

Creating the first Dynamic Code which exports a data set on page 14:

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

- ❖ The **Windows** version of **fpath_line** uses the specific drive letter:

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

- ❖ The **z/OS** version of the **fpath_line** 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® 9.4 Companion for z/OS:**

<http://support.sas.com/documentation/cdl/en/hosto390/68955/HTML/default/viewer.htm#p1t2wsrhr9x099n1h967cql2j3fm.htm>

- **SAS® 9.4 Companion for z/OS:**

<http://support.sas.com/documentation/cdl/en/hosto390/68955/HTML/default/viewer.htm#titlepage.htm>

Executing the 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:
 - **SAS(R) 9.4 Macro Language: Reference, Fourth Edition:**
<http://support.sas.com/documentation/cdl/en/mcrolref/67912/HTML/default/viewer.htm#n1q1527d51eivsn1ob5hnz0yd1hx.htm>
 - **SAS® 9.4 Companion for z/OS:**
<http://support.sas.com/documentation/cdl/en/hosto390/68955/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 27th Annual MWSUG 2016 Tools of the Trade Section Co-Chairs, **David Corliss**, **Sherry Zhou** (former), and **Deanna Schreiber-Gregory** (current) for graciously accepting our abstract and paper. In addition, we want to express our appreciation to the Conference Co-Chairs, **Richann Watson** (Academic Chair) and **Adrian Katschke** (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. And in conclusion, we want to give a shout out to our friend, **Charlie Shipp**, for his friendship and faithful service to the SAS World. You both 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 (2016), *Hands-On workshop: The Joinless Join ~ The Impossible Dream Come True; Expand the Power of Base SAS® and SAS® Enterprise Guide® in a New Way*, Proceedings of the 27th Annual MidWest SAS Users Group (MWSUG) 2016 Conference, The SASketeers, Des Moines, IA, USA.

Phelps, Kent ♥ Ronda Team (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*, Proceedings of the 26th Annual MidWest SAS Users Group (MWSUG) 2015 Conference, The SASketeers, Des Moines, IA, USA.

<http://www.mwsug.org/proceedings/2015/TT/MWSUG-2015-TT-05.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.

<http://www.mwsug.org/proceedings/2015/BI/MWSUG-2015-BI-11.pdf>

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. (2015), *SAS® 9.4 Companion for z/OS, Fifth Edition*; Cary, NC; SAS Institute Inc.

<http://support.sas.com/documentation/cdl/en/hosto390/68955/HTML/default/viewer.htm#hosto390whatsnew93.htm>

SAS Institute Inc. (2015), *SAS(R) 9.4 Macro Language: Reference, Fourth Edition*; Cary, NC; SAS Institute Inc.

<http://support.sas.com/documentation/cdl/en/mcrolref/67912/HTML/default/viewer.htm#mcrolrefwhatsnew94.htm>

SAS Institute Inc. (2015), *SAS(R) 9.4 Statements: Reference, Fourth Edition*; Cary, NC; SAS Institute Inc.

<http://support.sas.com/documentation/cdl/en/lestmtsref/68024/HTML/default/viewer.htm#p00hgx3x8lwivcn1f0e9axziw57y.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.