

Base SAS® & SAS® Enterprise Guide®
Automate Your SAS® World with Dynamic Code ~ Forwards & Backwards
Your newest BFF (Best Friend Forever) in SAS

Kent ♥ Ronda Team Phelps ~ The SASketeers ~ All for SAS & SAS for All!
Illuminator Coaching, Inc. ~ Des Moines, Iowa

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 re-creates the static code, and then executes the re-created static code automatically. Our workshop highlights the powerful partnership that occurs when dynamic code is creatively combined with a dynamic FILENAME statement, the SET INDSNAME option, a macro variable, and the CALL EXECUTE command within one SAS Enterprise Guide Program node.

You have the exciting opportunity to learn how to design dynamic code forwards and backwards to re-create static code while automatically changing the year ~ as 1,574 time-consuming manual steps are amazingly replaced with only one time-saving dynamic automated step. We invite you to attend our Dynamic Code Hands-On Workshop, in which we detail the UNIX and Microsoft Windows syntax for our project example and introduce you to your newest BFF (Best Friend Forever) in SAS. (Please see the appendixes to review additional starting-point information about the syntax for IBM 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 we like to say, SAS enables you to creatively program **Smarter And Smarter**. However, as remarkable as it is, SAS 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 increasingly 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 design a SAS Enterprise Guide program node to:

- ❖ Transform a static FILENAME statement into a dynamic FILENAME statement to obtain a Directory Listing of files from a folder on 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.
 - The dynamic code in this workshop re-creates static code which is executable code that never changes and always runs exactly the same way.
 - The dynamic code will be stored in a variable in a SAS data set.
- ❖ Execute the dynamic code automatically with no manual processing or intervention.

The SAS project in this workshop demonstrates:

The Power To Know through a dynamic FILENAME statement

The Power To Transform static code into dynamic code using the SET INDSNAME option and a macro variable

The Power To Execute dynamic code automatically using the CALL EXECUTE command

We invite you to journey with us as we share how

Dynamic Code

can become your BFF 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 SET INDSNAME 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. This project was prompted by a business need to make the research and analysis of vital variables from 14 years of weekly snapshot data sets more efficient. 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.

Project Requirements:

- ❖ Extract vital variables from 52 weekly data sets per year for 14 years (2006-2019) from a folder on the server 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 data sets per year to create 14 yearly data sets.
- ❖ Export the 14 appended yearly data sets back to the folder on the server.

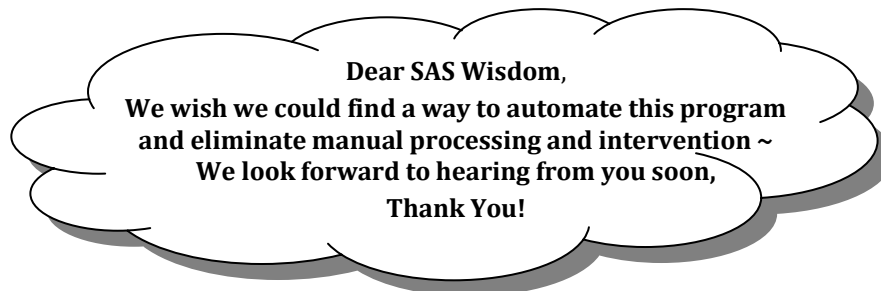
Since SAS Enterprise Guide was being used to design this project, the first decision to make was, “Should the program be designed with Graphical User Interface (GUI) and/or a program node?”

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

- ❖ What will it take to manually add 52 weekly 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 ~ 13 more times ~ while running the program for the 14 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 **1,574 manual steps** required to run the program for the 14 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.

Our first step was to revise the previous programming questions:

- ❖ What will it take to automatically create one DATA step to read and append 52 data sets together?
- ❖ What will it take to automatically extract vital variables in one DATA step?
- ❖ What will it take to automatically enter the derived value of the Load_Date variable in one 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 one DATA step ~ 13 more times ~ while running the program for the 14 year timeframe?

☺ **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.*

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 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 cover 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 Detour** ☺ ~ 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 in order to derive the value of a variable from the filenames of the files being read.

Obstacle Bump ~ A dynamic FILENAME statement can be used to obtain a Directory Listing that can be used to transform static code into dynamic code that automatically re-creates the static code; however, we determined the same dynamic FILENAME statement could not be used again within the re-created 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 each input observation as it is read!"

☺ **Happy Accident Detour** ☺ ~ 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 SET statement using the SET INDSNAME option.

Learning this information enabled us to design a program to use:

- ❖ A dynamic FILENAME statement to obtain one Directory Listing for 14 years of the filenames of the 52 weekly data sets per year from a folder on the server.
- ❖ The Directory Listing to transform Extract, Append, and Export static code into dynamic code that automatically re-creates 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 SET INDSNAME Option.
 - Append the 52 weekly data sets per year to create 14 yearly data sets.
 - Export the 14 appended yearly data sets back to the folder on the server.

Once the program has run, the re-created Extract, Append, and Export static code can be run **manually** by copying and pasting the code into another program node. Please note 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

Obstacle Bump ~ After we determined how to transform a static FILENAME statement into a dynamic FILENAME statement to obtain a Directory Listing to transform Extract, Append, and Export static code into dynamic code that automatically re-creates the static code ~ a very important question arose, "Is there also a way to execute the re-created 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 Detour** ☺ ~ It only took one more hopeful Google search to find a White Paper titled *CALL EXECUTE: A Powerful Data Management Tool* which revealed that a CALL EXECUTE command actually existed!

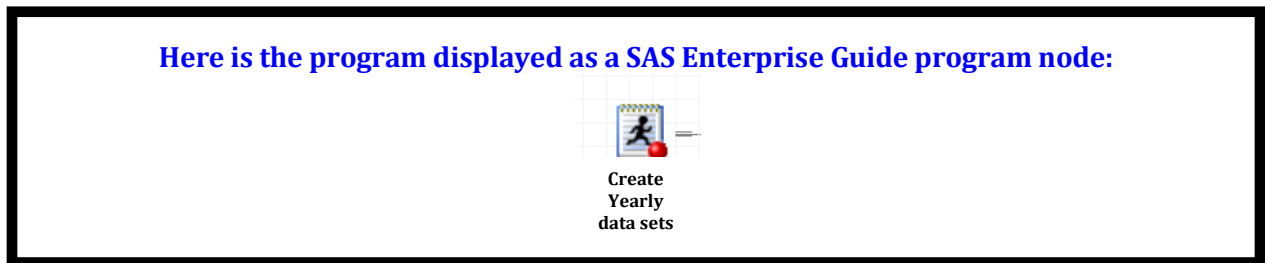
Here is a brief overview of the White Paper:

- ❖ 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.

O! The joy... of sweet success ~ when we discovered that the CALL EXECUTE command can execute the re-created static code automatically ~ and thus enable us to **COMPLETELY** automate this project!



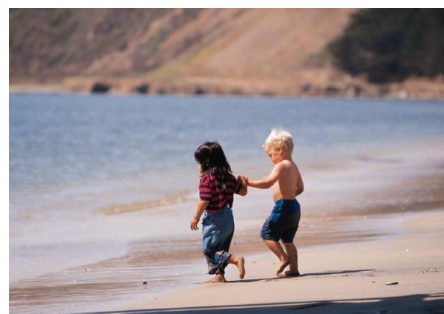
SAS Illumination ~ we will use a dynamic FILENAME statement to obtain a Directory Listing to transform Extract, Append, and Export static code into dynamic code that automatically re-creates the static code, and then use the CALL EXECUTE command to execute the static code automatically without any manual processing or intervention!



As you can see from this SAS quest, it pays to listen to your SAS intuition. Our determination to overcome obstacles ~ and 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, efficiency, and automation 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 workshop details the UNIX and Microsoft Windows syntax for our project example. Please refer to your specific Operating System (e.g. UNIX, Windows, or IBM z/OS) Manual, Installation Configuration, and/or in-house Technical Support for further guidance in how to create the SAS code presented. See Appendix A for additional starting-point information regarding the syntax for IBM z/OS.

The following example highlights how to transform a static FILENAME statement into a dynamic FILENAME statement to obtain a Directory Listing of the 52 weekly data sets for each year from a folder on the server.




This code will obtain and store the Directory Listing:



```
FILENAME indata '/SMILEY/file*.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 `file*.sas7bdat` pattern from the `SMILEY` folder on the server and store it in a data set.

These are the 3 weekly data sets being processed in our example:

 file20180105.sas7bdat
 file20180112.sas7bdat
 file20190104.sas7bdat

- ❖ Each must follow the same pattern `fileYYYYMMDD.sas7bdat` (See Appendix B for code to create).

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

- ❖ This contains each Special Person, Number, and Code for employees of the ☺ Smiley Company ☺.

Creating a dynamic FILENAME statement:

```
FILENAME indata '/SMILEY/file*.sas7bdat';
```

- ❖ The **FILENAME** statement assigns **indata** as a file reference (fileref) to the folder and file pattern.
- ❖ The asterisk within the file pattern **file*.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 to 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 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 each path and filename 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 for the Directory Listing 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 '/SMILEY/file*.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	/SMILEY/file20180105.sas7bdat
2	/SMILEY/file20180112.sas7bdat
3	/SMILEY/file20190104.sas7bdat

- ❖ Next 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 SET INDSNAME Option and a Macro Variable

The following example highlights 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 data sets and combine them with a Load_Date variable (derived from the filenames of the data sets), how to Append the 52 weekly data sets to create a yearly data set, and how to Export the yearly data set back to the folder on the server.

This is the original Extract static code for weeks 1 and 3:

```
DATA file_final_20180105;  
  SET '/SMILEY/file20180105.sas7bdat';  
  FORMAT Load_Date date9.; Load_Date = '05JAN2018'd;  
  KEEP Special_Person Special_Number Load_Date;  
RUN;
```

```
DATA file_final_20190104;  
  SET '/SMILEY/file20190104.sas7bdat';  
  FORMAT Load_Date date9.; Load_Date = '04JAN2019'd;  
  KEEP Special_Person Special_Number Load_Date;  
RUN;
```

- ❖ Each weekly **DATA** step creates a `file_final_YYYYMMDD` data set with the `Load_Date` variable derived and formatted as a SAS date (`date9`) from the `YYYYMMDD` create date of the input data set.
- ❖ A **KEEP** statement keeps the `Special_Person`, `Special_Number`, and derived `Load_Date`.

Here is the Append code beginning to be combined with the Export code:

```
DATA '/SMILEY/final_2018.sas7bdat' '/SMILEY/final_2019.sas7bdat';  
  SET file_final_20180105.sas7bdat  
      file_final_20180112.sas7bdat  
      file_final_20190104.sas7bdat;  
RUN;
```

- ❖ The Append **DATA** step creates `final_YYYY` data sets with `YYYY` matching each input data set year.
- ❖ Each of the `file_final_YYYYMMDD` data sets are **SET** as data sets one after another.

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

```
DATA '/SMILEY/final_2018.sas7bdat' '/SMILEY/final_2019.sas7bdat';  
  SET '/SMILEY/file20180105.sas7bdat'  
      '/SMILEY/file20180112.sas7bdat'  
      '/SMILEY/file20190104.sas7bdat';  
  FORMAT Load_Date date9.; Load_Date = '05JAN2018'd;  
  KEEP Special_Person Special_Number Load_Date;  
RUN;
```

- ❖ This Append **SETs** the 52 original data sets as inputs thereby eliminating the 52 Extract **DATA** steps.
- ❖ **Question:** Since the dynamic `FILENAME` statement 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 `Load_Date`?
- ❖ **Question:** How do we determine which output data set each input data set should be written to?

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

```
DATA '/SMILEY/final_2018.sas7bdat' '/SMILEY/final_2019.sas7bdat';
  LENGTH Current_File $100;
  SET '/SMILEY/file20180105.sas7bdat'
    '/SMILEY/file20180112.sas7bdat'
    '/SMILEY/file20190104.sas7bdat'
    INDSNAME=Current_File;
  FORMAT Load_Date date9.;
  KEEP Special_Person Special_Number Load_Date;
  Load_Year = SUBSTR(Current_File,13,4);
  Load_Date = MDY(INPUT(SUBSTR(Current_File,17,2),2.),
                 INPUT(SUBSTR(Current_File,19,2),2.),
                 INPUT(Load_Year,4.));
  SELECT(Load_Year);
    WHEN('2018') OUTPUT '/SMILEY/final_2018.sas7bdat';
    WHEN('2019') OUTPUT '/SMILEY/final_2019.sas7bdat';
  END;
RUN;
```

- ❖ Create a variable `Current_File` with a `LENGTH` long enough for each input data set name and path.
- ❖ Place the `LENGTH` before the `SET` statement so the complete data set name and path are captured.
- ❖ Add `INDSNAME=Current_File` to the end of the `SET` statement so `Current_File` 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_File`) into `Load_Year` and `Load_Date`.
- ❖ Use `SELECT(Load_Year)` to write each observation to the correct `OUTPUT` data set by `Load_Year`.

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

```
DATA '/SMILEY/final_2018.sas7bdat' '/SMILEY/final_2019.sas7bdat';
  LENGTH Current_File Prior_File $100;
  SET '/SMILEY/file20180105.sas7bdat'
    '/SMILEY/file20180112.sas7bdat'
    '/SMILEY/file20190104.sas7bdat'
    INDSNAME=Current_File;
  FORMAT Load_Date date9.;
  KEEP Special_Person Special_Number Load_Date;
  RETAIN Load_Date Load_Year;
  IF Current_File NE Prior_File THEN
    DO;
      File_Load_Year = SUBSTR(Current_File,13,4);
      IF Load_Year NE File_Load_Year THEN Load_Year = File_Load_Year;
      Load_Date = MDY(INPUT(SUBSTR(Current_File,17,2),2.),
                    INPUT(SUBSTR(Current_File,19,2),2.),
                    INPUT(Load_Year,4.));
      Prior_File = Current_File;
    END;
  SELECT(Load_Year);
    WHEN('2018') OUTPUT '/SMILEY/final_2018.sas7bdat';
    WHEN('2019') OUTPUT '/SMILEY/final_2019.sas7bdat';
  END;
RUN;
```

- ❖ Add `Prior_File`, `IF Current_File NE Prior_File THEN`, and `Prior_File = Current_File` to assign `Load_Date` with data set changes, and add `File_Load_Year` and `IF Load_Year NE File_Load_Year THEN` to assign `Load_Year = File_Load_Year` with data set year changes.
- ❖ Add `RETAIN Load_Date Load_Year` statement to retain `Load_Date` and `Load_Year`.

Begin to transform static code into dynamic code using quotes and timing:

```
FILENAME indata '/SMILEY/file*.sas7bdat';
DATA path_list_files;
  LENGTH SAS_data_set_and_path fpath $100 dyncode $10000;
  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;
        dyncode = CATS(
DATA '/SMILEY/final_2018'.sas7bdat'
'/SMILEY/final_2019'.sas7bdat';
LENGTH Current_File Prior_File $100;
SET '/SMILEY/file20180105.sas7bdat'
'/SMILEY/file20180112.sas7bdat'
'/SMILEY/file20190104.sas7bdat' INDSNAME=Current_File;
FORMAT Load_Date date9.;
KEEP Special_Person Special_Number Load_Date;
RETAIN Load_Date Load_Year;
IF Current_File NE Prior_File THEN
  DO;
    File_Load_Year = SUBSTR(Current_File,13,4);
    IF Load_Year NE File_Load_Year THEN Load_Year = File_Load_Year;
    Load_Date = MDY(INPUT(SUBSTR(Current_File,17,2),2.),
                    INPUT(SUBSTR(Current_File,19,2),2.),
                    INPUT(Load_Year,4.));
    Prior_File = Current_File;
  END;
  SELECT(Load_Year);
  WHEN('2018') OUTPUT '/SMILEY/final_2018'.sas7bdat';
  WHEN('2019') OUTPUT '/SMILEY/final_2019'.sas7bdat';
END;
RUN;);
OUTPUT;
END;
RUN;
```

Use quotation marks to surround the static code and what changes within:

- ❖ Create a variable `dyncode` long enough to contain the concatenation with spaces removed (`CATS`) of the static code in `quotation marks`.
- ❖ 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.
- ❖ Surround the years of the output data sets with `double quotes and commas` because they will be derived from the years of the input data sets.
- ❖ Surround the names of the input data sets with `double quotes and commas` because they will be derived from the names of the input data sets.

Identify the timing of changes with each observation of static code:

- ❖ The `DATA`, `LENGTH`, and `SET` statements will begin to be derived during the first observation and will be updated with each input data set name and output data set year ~ **forwards and backwards**.
- ❖ The `/SMILEY/file20180105.sas7bdat` through `/SMILEY/file20190104.sas7bdat` input data set names and the output data set years `2018` and `2019` will be derived from each observation of `fpath`.
- ❖ The `INDSNAME=Current_File;` through `RUN;);` statements will occur at the end after all observations of the Directory Listing have been read.

Code for the timing of changes with each observation of static code:

```

FILENAME indata '/SMILEY/file*.sas7bdat';
DATA path_list_files;
  LENGTH SAS_data_set_and_path fpath $100 dyncode $10000
         dyncode_data_outputs dyncode_set_inputs dyncode_when_outputs $1000;
  RETAIN fpath fpath_year dyncode_data_outputs dyncode_set_inputs
         dyncode_when_outputs;
  INFILE indata TRUNCOVER FILENAME=SAS_data_set_and_path;
  INPUT;

  IF _N_ = 1 THEN
    DO;
      dyncode_data_outputs = CATS("DATA ");
      dyncode_set_inputs =
        CATS("; LENGTH Current_File Prior_File $100; SET ");
    END;

  IF fpath NE SAS_data_set_and_path THEN
    DO;
      fpath = SAS_data_set_and_path;
      dyncode_set_inputs = CATS(dyncode_set_inputs)||
        " '||CATS(fpath)||' ";

      data_set_year = SUBSTR(fpath,13,4);
      IF fpath_year NE data_set_year THEN
        DO;
          fpath_year = data_set_year;
          dyncode_data_outputs = CATS(dyncode_data_outputs)||
            " '||CATS("/SMILEY/final_",
              data_set_year,".sas7bdat'");
          dyncode_when_outputs = CATS(dyncode_when_outputs)||
            " ||CATS("WHEN('",data_set_year,
              "') OUTPUT '/SMILEY/final_",
              data_set_year,".sas7bdat'");
        END;
    END;
  END;

```

- ❖ Add `fpath_year`, `dyncode_data_outputs`, `dyncode_set_inputs` and `dyncode_when_outputs` variables to the `RETAIN` and `LENGTH` statements to determine changes in the year and to enable each piece of the `DATA` step to be concatenated until the result is the complete dynamic code.
- ❖ Add `IF _N_ = 1 THEN` to begin the derivation of `dyncode_data_outputs` to be the `DATA` part of the `DATA` step. This derivation will continue with each changing year of the input data sets.
- ❖ `IF _N_ = 1 THEN` also begins the derivation of `dyncode_set_inputs` to be the `LENGTH` and `SET` part of the `DATA` step. This derivation will continue with each changing input data set.
- ❖ Add `IF fpath NE SAS_data_set_and_path THEN` to assign `fpath = SAS_data_set_and_path`; only when the first observation of each new input data set is read.
- ❖ Assign `dyncode_set_inputs` to itself from either the `IF _N_ = 1 THEN` logic or the previous `IF fpath NE SAS_data_set_and_path THEN` logic, remove the beginning and ending spaces `CATS(dyncode_set_inputs)` and concatenate `||` it to each input data set and path `CATS(fpath)` surrounded by single quotes `'`.
- ❖ Assign `data_set_year = SUBSTR(fpath,13,4)` to the year of the input data set.
- ❖ Add `IF fpath_year NE data_set_year THEN` to update `dyncode_data_outputs` to include each new output data set with each `data_set_year` in the `DATA` line.
- ❖ `IF fpath_year NE data_set_year THEN` also updates `dyncode_when_outputs` to include the new year and output data set with each `data_set_year` in the `SELECT WHEN OUTPUT` statement.

Code for the timing of changes after the final observation of static code:

```

FILENAME indata '/SMILEY/file*.sas7bdat';
DATA path_list_files;
  LENGTH SAS_data_set_and_path fpath $100 dyncode $10000
        dyncode_data_outputs dyncode_set_inputs dyncode_when_outputs $10000;
  RETAIN fpath fpath_year dyncode_data_outputs dyncode_set_inputs
        dyncode_when_outputs;
  INFILE indata TRUNCOVER FILENAME=SAS_data_set_and_path END=DONE;
  INPUT;
  . . .
  IF DONE THEN
    DO;
      dyncode = CATS(dyncode_data_outputs, dyncode_set_inputs) ||
        " INDSNAME=Current_File;
      FORMAT Load_Date date9.;
      KEEP Special_Person Special_Number Load_Date;
      RETAIN Load_Date Load_Year;
      IF Current_File NE Prior_File THEN
        DO;
          File_Load_Year = SUBSTR(Current_File,13,4);
          IF Load_Year NE File_Load_Year THEN Load_Year = File_Load_Year;
          Load_Date = MDY(INPUT(SUBSTR(Current_File,17,2),2.),
                        INPUT(SUBSTR(Current_File,19,2),2.),
                        INPUT(Load_Year,4.));
          Prior_File = Current_File;
        END;
      SELECT(Load_Year); " || dyncode_when_outputs ||
      " END;
    RUN;";
    OUTPUT;
  END;
RUN;

```

- ❖ Add **END=DONE** to the **INFILE** statement so **DONE** will be set to True and the **IF DONE THEN** will execute after the last observation is read from the last input data set.
- ❖ Assign **dyncode** to the concatenation of **dyncode_data_outputs** (**DATA** statement and output data sets), **dyncode_set_inputs** (**SET** statement and data sets), the static code beginning with **INDSNAME** and ending with **SELECT(Load_Year);**, **dyncode_when_outputs** (**SELECT WHEN OUTPUT** statements), and the final **END** and **RUN** statements.

Assign a macro variable to the current file month, day, and year:

```

%LET Current_File_MDY = INPUT(SUBSTR(Current_File,17,2),2.),
                        INPUT(SUBSTR(Current_File,19,2),2.),
                        INPUT(Load_Year,4.);
...
      IF Current_File NE Prior_File THEN
...
          Load_Date = MDY(&Current_File_MDY);

```

- ❖ Add the **%LET** to assign a new macro variable called **Current_File_MDY** to the three lines of code in the parenthesis of the **MDY** function of the **Load_Date** assignment.
- ❖ Replace the code in the parenthesis of the **MDY** function with the macro **&Current_File_MDY** ~ please note that double quotes " already surround this section of code in the previous code box, so no further changes are necessary.

This is the dynamic code that recreates the original static code:

```

%LET Current_File_MDY = INPUT(SUBSTR(Current_File,17,2),2.),
                          INPUT(SUBSTR(Current_File,19,2),2.),
                          INPUT(Load_Year,4.);
FILENAME indata '/SMILEY/file*.sas7bdat';
DATA path_list_files;
  LENGTH SAS_data_set_and_path fpath $100 dyncode $10000
         dyncode_data_outputs dyncode_set_inputs dyncode_when_outputs $1000;
  RETAIN fpath fpath_year dyncode_data_outputs dyncode_set_inputs
         dyncode_when_outputs;
  INFILE indata TRUNCOVER FILENAME=SAS_data_set_and_path END=DONE;
  INPUT;
  IF _N_ = 1 THEN
    DO;
      dyncode_data_outputs = CATS("DATA ");
      dyncode_set_inputs =
        CATS("; LENGTH Current_File Prior_File $100; SET ");
    END;
  IF fpath NE SAS_data_set_and_path THEN
    DO;
      fpath = SAS_data_set_and_path;
      dyncode_set_inputs = CATS(dyncode_set_inputs||
        " "||CATS(fpath)||" " );
      data_set_year = SUBSTR(fpath,13,4);
      IF fpath_year NE data_set_year THEN
        DO;
          fpath_year = data_set_year;
          dyncode_data_outputs = CATS(dyncode_data_outputs||
            " "||CATS("/SMILEY/final_",
              data_set_year,".sas7bdat'");
          dyncode_when_outputs = CATS(dyncode_when_outputs||
            " "||CATS("WHEN('",data_set_year,
              "') OUTPUT '/SMILEY/final_",
              data_set_year,".sas7bdat'");
        END;
    END;
  IF DONE THEN
    DO;
      dyncode = CATS(dyncode_data_outputs,dyncode_set_inputs)||
        " INDSNAME=Current_File;
      FORMAT Load_Date date9.;
      KEEP Special_Person Special_Number Load_Date;
      RETAIN Load_Date Load_Year;
      IF Current_File NE Prior_File THEN
        DO;
          File_Load_Year = SUBSTR(Current_File,13,4);
          IF Load_Year NE File_Load_Year THEN Load_Year = File_Load_Year;
          Load_Date = MDY(&Current_File_MDY);
          Prior_File = Current_File;
        END;
      SELECT(Load_Year); "||dyncode_when_outputs||
        " END;
    RUN;";
  OUTPUT;
END;
RUN;

```

- ❖ We added the assignment of the `Current_File_MDY` macro variable to the beginning of the code and the application of this macro variable `&Current_File_MDY` to the assignment of the `Load_Date` variable.

THE POWER TO EXECUTE

Dynamic Code Automatically Using the CALL EXECUTE Command

After transforming 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 will execute the code automatically.

This is the dynamic code which recreates and runs the original static code:

```

%LET Current_File_MDY = INPUT(SUBSTR(Current_File,17,2),2.),
                          INPUT(SUBSTR(Current_File,19,2),2.),
                          INPUT(Load_Year,4.);
FILENAME indata '/SMILEY/file*.sas7bdat';
DATA path_list_files;
  LENGTH SAS_data_set_and_path fpath $100 dyncode $10000
        dyncode_data_outputs dyncode_set_inputs dyncode_when_outputs $1000;
  RETAIN fpath fpath_year dyncode_data_outputs dyncode_set_inputs
        dyncode_when_outputs;
  INFILE indata TRUNCOVER FILENAME=SAS_data_set_and_path END=DONE;
  INPUT;
  IF _N_ = 1 THEN
    DO;
      dyncode_data_outputs = CATS("DATA ");
      dyncode_set_inputs =
        CATS(" ; LENGTH Current_File Prior_File $100; SET ");
    END;
  IF fpath NE SAS_data_set_and_path THEN
    DO;
      fpath = SAS_data_set_and_path;
      dyncode_set_inputs = CATS(dyncode_set_inputs)||
        " '||CATS(fpath)||' ";
      data_set_year = SUBSTR(fpath,13,4);
      IF fpath_year NE data_set_year THEN
        DO;
          fpath_year = data_set_year;
          dyncode_data_outputs = CATS(dyncode_data_outputs)||
            " '||CATS("/SMILEY/final_",
              data_set_year,".sas7bdat");
          dyncode_when_outputs = CATS(dyncode_when_outputs)||
            " ||CATS("WHEN('",data_set_year,
              "') OUTPUT '/SMILEY/final_",
              data_set_year,".sas7bdat'");
        END;
    END;
  IF DONE THEN
    DO;
      dyncode = CATS(dyncode_data_outputs,dyncode_set_inputs)||
        " INDSNAME=Current_File;
      FORMAT Load_Date date9.;
      KEEP Special_Person Special_Number Load_Date;
      RETAIN Load_Date Load_Year;
      IF Current_File NE Prior_File THEN
        DO;
          File_Load_Year = SUBSTR(Current_File,13,4);
          IF Load_Year NE File_Load_Year THEN Load_Year = File_Load_Year;
          Load_Date = MDY(&Current_File_MDY);
          Prior_File = Current_File;
        END;
      SELECT (Load_Year); ||dyncode_when_outputs||
      " END;
  RUN;";
  OUTPUT;
  CALL EXECUTE (dyncode);
END;
RUN;

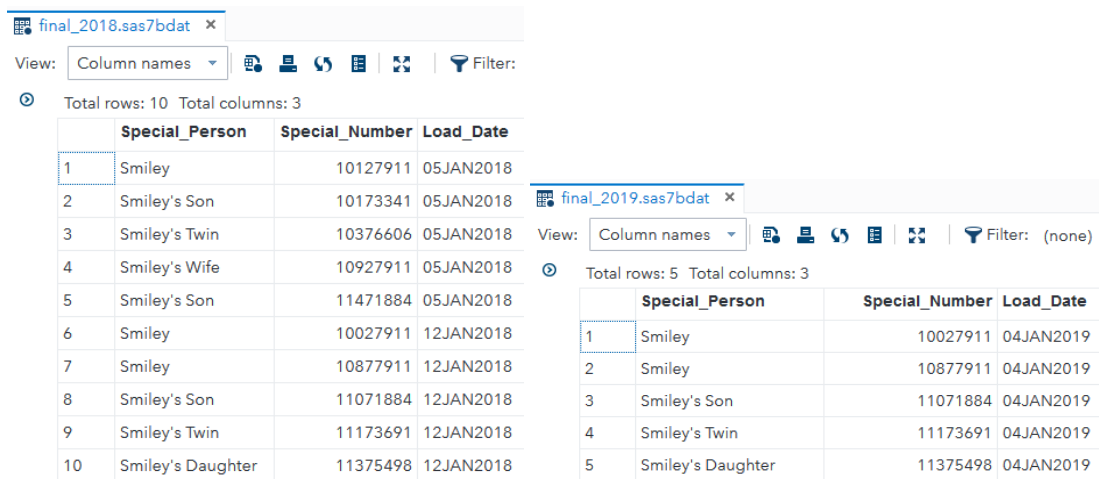
```

❖ The CALL EXECUTE command executes the contents of the dyncode variable which was just created.

Here is the code with the dynamic code resolved as the original static code:

```
DATA '/SMILEY/final_2018.sas7bdat' '/SMILEY/final_2019.sas7bdat';
LENGTH Current_File Prior_File $100;
SET '/SMILEY/file20180105.sas7bdat'
    '/SMILEY/file20180112.sas7bdat'
    '/SMILEY/file20190104.sas7bdat'
    INDSNAME=Current_File;
FORMAT Load_Date date9.;
KEEP Special_Person Special_Number Load_Date;
RETAIN Load_Date Load_Year;
IF Current_File NE Prior_File THEN
DO;
    File_Load_Year = SUBSTR(Current_File,13,4);
    IF Load_Year NE File_Load_Year THEN Load_Year = File_Load_Year;
    Load_Date = MDY(INPUT(SUBSTR(Current_File,17,2),2.),
                    INPUT(SUBSTR(Current_File,19,2),2.),
                    INPUT(Load_Year,4.));
    Prior_File = Current_File;
END;
SELECT (Load_Year);
WHEN('2018') OUTPUT '/SMILEY/final_2018.sas7bdat';
WHEN('2019') OUTPUT '/SMILEY/final_2019.sas7bdat';
END;
RUN;
```

❖ This is the result of fully executing `dyncode` in the `path_list_files` data set:



	Special_Person	Special_Number	Load_Date
1	Smiley	10127911	05JAN2018
2	Smiley's Son	10173341	05JAN2018
3	Smiley's Twin	10376606	05JAN2018
4	Smiley's Wife	10927911	05JAN2018
5	Smiley's Son	11471884	05JAN2018
6	Smiley	10027911	12JAN2018
7	Smiley	10877911	12JAN2018
8	Smiley's Son	11071884	12JAN2018
9	Smiley's Twin	11173691	12JAN2018
10	Smiley's Daughter	11375498	12JAN2018

	Special_Person	Special_Number	Load_Date
1	Smiley	10027911	04JAN2019
2	Smiley	10877911	04JAN2019
3	Smiley's Son	11071884	04JAN2019
4	Smiley's Twin	11173691	04JAN2019
5	Smiley's Daughter	11375498	04JAN2019

❖ Please note that **Windows** users will need to add the specific drive letter to the path and reverse the slashes from / to \ as shown below:

```
/* UNIX */ FILENAME indata '/SMILEY/file*.sas7bdat';
/* Windows */ FILENAME indata 'c:\SMILEY\file*.sas7bdat';

/* UNIX */ DATA '/SMILEY/final_2018.sas7bdat' '/SMILEY/final_2019.sas7bdat';
/* Windows */ DATA 'c:\SMILEY\final_2018.sas7bdat' 'c:\SMILEY\final_2019.sas7bdat';
```

☺ 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 SET INDSNAME option and a macro variable which leads to **The Power To Execute** dynamic code automatically using the CALL EXECUTE command. {☺Try saying that statement really fast for fun☺} You have seen how 1,574 time-consuming manual steps, including changing the year, are amazingly replaced with only one 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 and automation 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 empower those around you! Always remember, your contributions make a positive impact in the world. Plan on coming back to the MWSUG Regional 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.



*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

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 grand and noble 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 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 ☺



ACKNOWLEDGMENTS

We want to thank the 30th Annual MWSUG 2019 **Hands-On-Workshop Chair, Jay Iyengar**, for accepting our abstract and paper, and we want to express our appreciation to the **Conference Co-Chairs, Jessica Chen (Academic Chair) and Adrian Katschke (Operations Chair)**, the Executive Committee and Conference Leaders, and SAS Institute for their diligent efforts in organizing this illuminating conference.

You inspire us to share what we are learning and we hope to be a light of encouragement to you as well ~ Your friends, Kent ♥ Ronda **Team Phelps** ~ **The SASketeers** ~ **All for SAS & SAS for All!** ~ Illuminator Coaching, Inc.

MEET THE AUTHORS

Writing is a permanent legacy.

John C. Maxwell

Kent ♥ Ronda Team Phelps are the co-founders of Illuminator Coaching, Inc., and *The SASketeers: All for SAS and SAS for All!* ~ they have co-authored 14 SAS White Papers ~ presented and co-presented at the MidWest SAS® Users Group (MWSUG) Regional Conference for the last 7 years, including 4 Hands-On Workshops ~ and co-presented at the SAS Global Forum (SGF) 2018 International Conference.

Kent wants to encourage and equip you to fulfill your life, career, and leadership potential as you build an enduring legacy of inspiration, excellence, and honor ~ SAS® Certified Professional Programmer Analyst Consultant ~ B.S. Electrical Engineering ~ serving for over 20 years as an essential bridge builder of consensus and quality programming to connect the needs of Business and IT ~ happily programmed in Base SAS® and SAS® Enterprise Guide® since 2007 with a strong focus on engineering innovative, efficient, and automated solutions.

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 ~ Writer & Coach ~ gifted in helping others to explore and express their hearts and minds through writing ~ served in the Banking and Insurance industries for 19 years.

CONTACT INFORMATION

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@IlluminatorCoaching.com

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

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.lexjansen.com/wuss/2012/55.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 Regional 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://support.sas.com/resources/papers/proceedings/proceedings/sugi30/027-30.pdf>

Phelps, Kent ♥ Ronda Team (2018), Hands-On Workshop (HOW): *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 29th Annual MidWest SAS Users Group (MWSUG) 2018 Regional Conference, The SASketeers ~ All for SAS and SAS for All! ~ Illuminator Coaching, Inc., Des Moines, IA, USA.

<http://www.mwsug.org/proceedings/2018/HW/MWSUG-2018-HW-98.pdf>

Phelps, Kent ♥ Ronda Team (2018), *Base SAS® and SAS® Enterprise Guide®: Automate Your SAS® World with Dynamic Code ~ Your Newest BFF (Best Friend Forever) in SAS*, Proceedings of SAS® Global Forum (SGF) 2018 International Conference, The SASketeers ~ All for SAS and SAS for All! ~ Illuminator Coaching, Inc., Des Moines, IA, USA.

<http://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2857-2018.pdf>

Phelps, Kent ♥ Ronda Team (2017), Hands-On Workshop (HOW): *Base SAS® and SAS® Enterprise Guide® ~ Automate Your SAS World With Dynamic Code; Your Newest BFF (Best Friend Forever) in SAS*, Proceedings of the 28th Annual MidWest SAS Users Group (MWSUG) 2017 Regional Conference, The SASketeers ~ All for SAS and SAS for All! ~ Illuminator Coaching, Inc., Des Moines, IA, USA.

<http://www.mwsug.org/proceedings/2017/HW/MWSUG-2017-HW03.pdf>

Phelps, Kent ♥ Ronda Team (2016), *Base SAS® and SAS® Enterprise Guide® ~ Automate Your SAS World With Dynamic Code; Your Newest BFF (Best Friend Forever) in SAS*, Proceedings of the 27th Annual MidWest SAS Users Group (MWSUG) 2016 Regional Conference, The SASketeers ~ All for SAS and SAS for All! ~ Illuminator Coaching, Inc., Des Moines, IA, USA.

<http://www.mwsug.org/proceedings/2016/TT/MWSUG-2016-TT11.pdf>

Phelps, Kent ♥ Ronda Team (2016), Hands-On Workshop (HOW): *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 Regional Conference, The SASketeers ~ All for SAS and SAS for All! ~ Illuminator Coaching, Inc., Des Moines, IA, USA.

<http://www.mwsug.org/proceedings/2016/HW/MWSUG-2016-HW03.pdf>

SAS Institute Inc. (2016), *SAS® 9.4 Companion for z/OS, Sixth Edition*; Cary, NC; SAS Institute Inc.

<http://documentation.sas.com/api/docsets/hosto390/9.4/content/hosto390.pdf?locale=en#nameddest=titlepage>

SAS Institute Inc. (2016), *SAS® 9.4 Macro Language: Reference, Fifth Edition*; Cary, NC; SAS Institute Inc.
<http://documentation.sas.com/api/docsets/mcrolref/9.4/content/mcrolref.pdf?locale=en#nameddest=titlepage>

SAS Institute Inc. (2016), *SAS® 9.4 DATA Step Statements: Reference*; Cary, NC; SAS Institute Inc.
<http://documentation.sas.com/api/docsets/lestmtsref/9.4/content/lestmtsref.pdf?locale=en#nameddest=titlepage>

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>

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 Hands-On Workshop / Presentation / 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 company are/is 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, Microsoft Windows, or IBM z/OS) Manual, Installation Configuration, and/or in-house Technical Support for further guidance in how to create the SAS code presented.

APPENDIX A

Starting-Point Information About the Syntax for IBM z/OS

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

Creating the Dynamic FILENAME Statement:

```
FILENAME indata '/SMILEY/file*.sas7bdat';
```

- ❖ The z/OS 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://documentation.sas.com/?docsetId=hosto390&docsetTarget=n0yrspsthx1w5n1gyt6rgzh3qsu.htm&docsetVersion=9.4&locale=en>
 - **Accessing UNIX System Services Files from SAS® 9.4 Companion for z/OS:**
<http://documentation.sas.com/?docsetId=hosto390&docsetVersion=9.4&docsetTarget=n001udvg5mzcb1n1bhts48m1bal1.htm&locale=en>

Creating the first dynamic code which exports a data set:

```
dyncode_data_outputs = CATS("DATA '/SMILEY/file_",
```

- ❖ The z/OS **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://documentation.sas.com/?docsetId=hosto390&docsetVersion=9.4&docsetTarget=n1v0dy64syrm22n1o5kwd0qt6qd9.htm&locale=en>
 - **SAS® 9.4 Companion for z/OS:**
<http://documentation.sas.com/?docsetId=hosto390&docsetTarget=titlepage.htm&docsetVersion=9.4&locale=en>

Executing the CALL EXECUTE command:

```
CALL EXECUTE (dyncode) ;
```

- ❖ The z/OS **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® 9.4 Macro Language: Reference, Fourth Edition:**
<http://documentation.sas.com/?docsetId=mcrolref&docsetTarget=n1q1527d51eivsn1ob5hzn0yd1hx.htm&docsetVersion=9.4&locale=en>
 - **SAS® 9.4 Companion for z/OS:**
<http://documentation.sas.com/?docsetId=hosto390&docsetTarget=titlepage.htm&docsetVersion=9.4&locale=en>

APPENDIX B

The Code that Created the Data Sets for Our Project Example

```
DATA '/SMILEY/file20180105.sas7bdat'  
      '/SMILEY/file20180112.sas7bdat'  
      '/SMILEY/file20190104.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 '/SMILEY/file20180105.sas7bdat';  
  WHEN(_N_ LE 10) OUTPUT '/SMILEY/file20180112.sas7bdat';  
  OTHERWISE       OUTPUT '/SMILEY/file20190104.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  
;  
RUN;
```