

## **SAS® Commands PIPE and CALL EXECUTE;** *Dynamically Advancing from Strangers to Your Newest BFF (Best Friends Forever)*

Kent ♥ Ronda Team Phelps, The SASketeers, Des Moines, Iowa  
Kirk Paul Lafler, Software Intelligence Corporation, Spring Valley, California

### **ABSTRACT**

Communication is the basic foundation of all relationships, including our relationship with SAS and the Server/PC/Mainframe (S/P/M). There are times when we need to communicate with the S/P/M through the UNIX, Windows, or z/OS Operating System (OS). To communicate with the S/P/M we will ideally design our SAS programs to request, receive, and utilize data to automatically create and execute Dynamic Code.

Our presentation highlights the powerful partnership which occurs when the PIPE and CALL EXECUTE commands are combined with Dynamic Code and creatively used together within SAS Enterprise Guide® Base SAS® Program Nodes. You will have the opportunity to learn how **1,259** time-consuming Manual Steps are amazingly replaced with only **3** time-saving Dynamic Automated Steps. We invite you to attend our session where we detail the UNIX syntax for the PIPE and CALL EXECUTE commands and the Dynamic Code.

**We look forward to introducing you to your newest BFF (Best Friends Forever) in SAS.**

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

### **INTRODUCTION**



SAS is highly regarded around the world, and rightly so, as a powerful, intuitive, and flexible programming language. SAS stands for Statistical Analysis Software or, as we like to say, Smarter And Smarter. However, the SAS programming language, as amazing as it is, is not an island unto itself.

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

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

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



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

- ❖ Use Static Code to create a communication pipeline to request and receive data from the server.
  - Static Code is executable code which never changes and always runs exactly the same way.
- ❖ Utilize the data from the server to automatically create Dynamic Code.
  - Dynamic Code is executable code automatically created within a SAS data set, based upon parameters which can change, and therefore may or may not run exactly the same way.
- ❖ Execute the Dynamic Code automatically with no manual processing or intervention.

### The SAS Project in this presentation demonstrates:

**The Power To Know** through the PIPE command

**The Power To Create** Static Code which automatically creates Dynamic Code

**The Power To Execute** Dynamic Code automatically using the CALL EXECUTE command

We invite you to journey with us  
as we share how the PIPE and CALL EXECUTE commands  
were discovered and soon became  
**Best Friends Forever.**

### How PIPE and CALL EXECUTE Became Best Friends

☺ *A Tale of SAS Wis-h-dom* ☺

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

**Discovering the power** of combining the PIPE and CALL EXECUTE commands **was**, as Bob Ross, the well-known painter on PBS, so often said, “**A happy accident.**” When Bob was unable to paint something he had planned in a painting and had to paint something different, he referred to the detour as a Happy Accident. Likewise, when we start to search for one particular programming solution, which we may or may not find, we will often accidentally discover other creative ways to accomplish our Project Requirements.

Several discoveries occurred on a recent **SAS Quest** which we are eager to share with you through our project example. This project was prompted by a business need to greatly increase the efficiency of the research and analysis of vital variables from **11** years of weekly snapshot SAS data sets. The goal was to condense **572** weekly data sets to **11** yearly data sets. 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 snapshot data sets per year for **11** years (**2003-13**) and combine them with a Load\_Date variable (created from the Friday date value derived from the Filenames of the data sets) to create **572** new data sets.
- ❖ **Append** the **52** new data sets per year to create **11** yearly data sets.
- ❖ **Export** the **11** appended yearly data sets back to the folder on the server where the weekly snapshot data sets are stored.

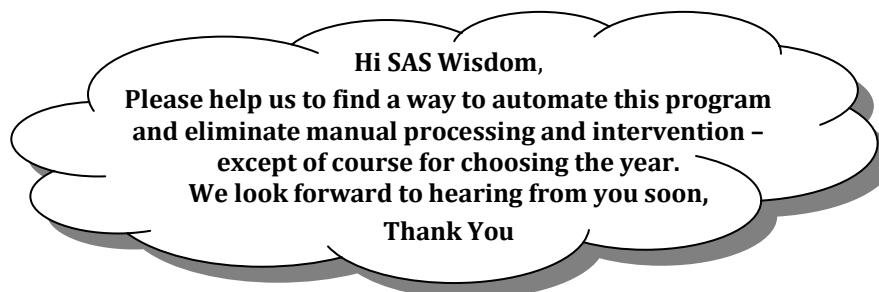
Since SAS Enterprise Guide was being used to design this project the first decision to make was, “To GUI or not to GUI?” In other words, should the program be designed using Graphical User Interface (GUI) and/or Base SAS Program Nodes?

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

- ❖ What will it take to **manually** add **52** weekly data sets to the project?
- ❖ What will it take to **manually** create **52** queries to select vital variables from **52** data sets?
- ❖ What will it take to **manually** enter the derived value of the Load\_Date variable in **52** queries?
- ❖ What will it take to **manually** append the **52** new data sets created by the **52** queries?
- ❖ What will it take to **manually** export the appended yearly data set back to the server?
- ❖ Once the program is designed, what will it take to **manually** swap **52** inputs and **manually** update the Load\_Date variable in **52** queries – **10** more times – while running the program for the **11** 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 **1,259 manual steps** required to run the program for the **11** year timeframe would be too manually intensive and prone to errors. As a result of this challenging realization, **SAS Intuition** said, “There must be a smarter, easier, and faster way to do this in SAS!” Thus, the following SAS Wish email was imagined:



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 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 seem overwhelming to figure out how to accomplish the requirements. Keep in mind that all we really need to do is take the first step – and the rest will soon follow.

### ☺ Maxwell's/Phelps'/Lafler's Law ☺

*Nothing is as hard as it looks;  
everything is more rewarding than you expect;  
and if anything can go right  
it will  
and at the best possible moment.*

#### Our first step was to revise the previous programming questions:

- ❖ What will it take to **automatically** create **52** DATA steps to read **52** data sets?
- ❖ What will it take to **automatically** extract vital variables in **52** DATA steps?
- ❖ What will it take to **automatically** enter the derived value of the Load\_Date variable in **52** DATA steps?
- ❖ What will it take to **automatically** append the **52** new data sets created by the **52** DATA steps?
- ❖ What will it take to **automatically** export the appended yearly data set back to the server?
- ❖ Once the program is designed, what will it take to **automatically** swap **52** inputs and **automatically** update the Load\_Date variable in **52** DATA steps – **10** more times – while running the program for the **11** year timeframe?

When the decision was made to automate this program a quest was undertaken to accomplish this grand and noble feat ☺. The next step was to find a way to design a **Dynamic INFILE 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 quickly led to an article titled *Using FILEVAR= To Read Multiple External Files in a DATA Step*.

#### Here is a brief overview of this article:

- ❖ The article explained different ways to use Dynamic INFILE Statements to automatically and sequentially read the content of multiple files.
- ❖ Unfortunately, the examples seemed to indicate that they cannot also derive the value of a variable from the Filenames of the files being read, and therefore could not fulfill one of the project requirements.
- ❖ ☺ **Happy Accident Alert** ☺ – A section titled *Reading All The Files From A Directory Using A Pipe*:
  - The PIPE command can be used in a Dynamic INFILE Statement to create a communication pipeline between a SAS program and the server through the OS to request and receive a Directory Listing of the Filenames from a folder.
  - The Directory Listing can then be utilized to read the content of the files while also deriving the value of a variable from the filenames of the files being read.

### Learning this information led to 3 programs being designed:

- ❖ **Program 1** – Design Static Code (including the PIPE command in a Dynamic INFILE Statement) to request, receive, and utilize one Directory Listing (per year for **11** years of the Filenames of the **52** weekly snapshot data sets) to automatically create Dynamic Code which will automatically **Extract** vital variables (from the data sets) and combine them with a Load\_Date variable (created from the Friday date value derived from the Filenames of the data sets) to create **52** new data sets per year for **11** years.
- ❖ **Program 2** – Design Static Code to utilize the Directory Listing to automatically create Dynamic Code which will automatically **Append** the **52** new data sets per year to create **11** yearly data sets.
- ❖ **Program 3** – Design Static Code to utilize the Directory Listing to automatically create Dynamic Code which will automatically **Export** the **11** appended yearly data sets back to the folder on the server where the weekly snapshot data sets are stored.

Once the **3** programs are run, the automatically created Dynamic Code can be run **manually** by copying and pasting the Dynamic Code into another Program Node. These **3** programs fulfill 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 design Static Code to request, receive, and utilize a Directory Listing to automatically create Dynamic Code, a very important question arose – **Is there also a way to automatically execute the Dynamic Code?** SAS Intuition spoke again, “There must be a way to call and execute a variable in a SAS data set containing a SAS DATA step.”

To our surprise, through another hopeful Google search, we discovered a White Paper titled **CALL EXECUTE: A Powerful Data Management Tool** which revealed that an actual CALL EXECUTE command already existed!

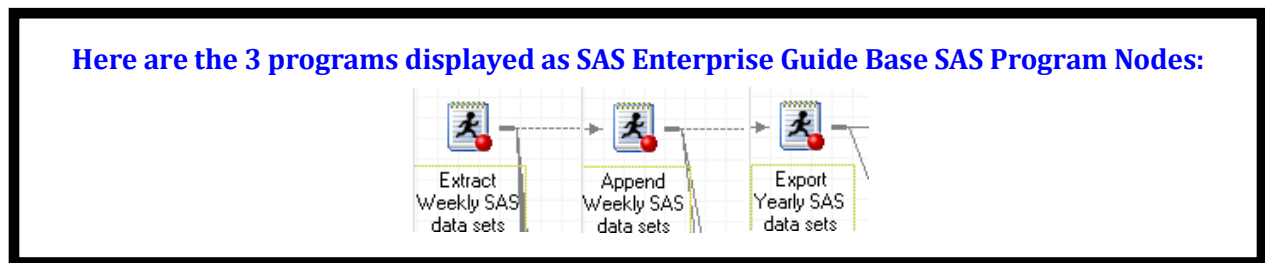
### Here is a brief overview of this White Paper:

- ❖ 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.
- ❖ ☺ **Happy Accident Alert** ☺ – CALL EXECUTE is all we need to execute the Dynamic Code automatically!



**This knowledge led to SAS Illumination** – The PIPE command will enable our program to instantly create a communication pipeline to the server through the OS. We will use this pipeline to send OS commands to request, receive, and utilize a Directory Listing to automatically create Dynamic Code which we will then execute automatically with the CALL EXECUTE command.

Combining the PIPE command with Dynamic Code and the CALL EXECUTE command enables the 3 SAS programs to automatically Extract, Append, and Export without any manual processing or intervention – except for choosing the year!



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

**On the next leg of our journey**

**we will walk you through a**

**step-by-step demonstration of**

**The Power To Know, Create, and Execute.**



## THE POWER TO KNOW Through the PIPE Command

**Disclaimer:** 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. Our project example details the UNIX syntax for the PIPE and CALL EXECUTE commands and the Dynamic Code. Please see **Appendix A** for starting point information regarding the syntax for Windows and z/OS.

The following examples highlight how to use the PIPE command to request and receive one Directory Listing of the Filenames of the **52** weekly data sets for the year **2013** from a folder on the server.

### How to request and receive a Directory Listing through the PIPE command:



```
FILENAME Inpipe PIPE 'ls /data/MWSUG/PIPE_CALL_EXECUTE/file2013*.sas7bdat';
DATA path_list_files;
  LENGTH fpath $100;
  INFILE Inpipe TRUNCOVER;
  INPUT fpath $100.;
  ...
RUN;
```

- ❖ This code creates a data set containing a Directory Listing of data sets following the `file2013*.sas7bdat` pattern from the `/data/MWSUG/PIPE_CALL_EXECUTE` folder on the server.
- ❖ Before walking through each line of code, we will first look at the data sets contained in this folder.
- ❖ Please see **Appendix B** for an alternative to the PIPE command.

### Here is a listing of the 7 weekly data sets being processed in our example:

```
file20130104.sas7bdat
file20130111.sas7bdat
file20130118.sas7bdat
file20130125.sas7bdat
file20130201.sas7bdat
file20130208.sas7bdat
file20130215.sas7bdat
```

- ❖ Notice how each of these data sets follow the same pattern of `fileYYYYMMDD.sas7bdat`.
- ❖ This Filename pattern will be essential in successfully creating Dynamic Code to Extract the data sets.
- ❖ Please see **Appendix C** for the code that creates these data sets.



### Here is the file20130104 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 data set contains each Special Person, Special Number, and Special Code for the employees of the 😊 Smiley Company 😊.
- ❖ Now it is time to explore the PIPE command and learn how it can help us to Extract these data sets.

### Creating a FILENAME statement containing the PIPE command:

```
FILENAME Inpipe PIPE 'ls /data/MWSUG/PIPE_CALL_EXECUTE/file2013*.sas7bdat';
```

- ❖ The **FILENAME** statement assigns **Inpipe** as a file reference (fileref) to the communication pipeline created by the **PIPE** command.
- ❖ The **PIPE** command sends an OS command – **ls** – to the server to request a List Contents:  
**'ls /data/MWSUG/PIPE\_CALL\_EXECUTE/file2013\*.sas7bdat'**
- ❖ The result is a Directory Listing received back through the pipeline by the **Inpipe** fileref.
- ❖ In summary, **FILENAME** assigns **Inpipe** to point to the Directory Listing via the **PIPE** command.

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

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

- ❖ The **DATA** statement creates an output data set called **path\_list\_files**.
- ❖ The **LENGTH** statement assigns a length of **100** characters to a variable called **fpath**.
- ❖ In summary, the **path\_list\_files** data set is created to contain the **100** character **fpath** variable.



## Preparing the Inpipe Fileref for use:

```
INFILE Inpipe TRUNCOVER;
```

- ❖ The **INFILE** statement assigns **Inpipe** (Directory Listing) to be read with the upcoming **INPUT** statement.
- ❖ The **TRUNCOVER** option tells SAS the input data may or may not be the same length.
- ❖ In summary, **INFILE** assigns **Inpipe** (Directory Listing) to be read with an **INPUT** of variable length.

## The INPUT of data begins:

```
INPUT fpath $100.;
```

- ❖ The **INPUT** statement reads the **INFILE Inpipe** (Directory Listing) one record at a time.
- ❖ The **fpath** variable stores up to **100** characters read from each record.
- ❖ In summary, **INPUT** reads the **INFILE Inpipe** (Directory Listing) one record at a time and stores up to **100** characters in the **fpath** variable.

## Here is how these statements look when combined with a RUN statement:

```
FILENAME Inpipe PIPE 'ls /data/MWSUG/PIPE_CALL_EXECUTE/file2013*.sas7bdat';  
DATA path_list_files;  
  LENGTH fpath $100;  
  INFILE Inpipe TRUNCOVER;  
  INPUT fpath $100.;;  
RUN;
```

## Here is the output data set created using the preceding statements:

	fpath
1	/data/MwSUG/PIPE_CALL_EXECUTE/file20130104.sas7bdat
2	/data/MwSUG/PIPE_CALL_EXECUTE/file20130111.sas7bdat
3	/data/MwSUG/PIPE_CALL_EXECUTE/file20130118.sas7bdat
4	/data/MwSUG/PIPE_CALL_EXECUTE/file20130125.sas7bdat
5	/data/MwSUG/PIPE_CALL_EXECUTE/file20130201.sas7bdat
6	/data/MwSUG/PIPE_CALL_EXECUTE/file20130208.sas7bdat
7	/data/MwSUG/PIPE_CALL_EXECUTE/file20130215.sas7bdat

- ❖ Next we will explore how the **fpath** variable is used to create Dynamic Code.

## THE POWER TO CREATE

### Static Code Which Automatically Creates Dynamic Code

The following examples highlight how to create Static Code which automatically creates Dynamic Code to automatically **Extract** vital variables from **52** weekly data sets and combine them with a Load\_Date variable (created from the Friday date value derived from the Filenames of the data sets) to create **52** new data sets.

#### How to Extract vital variables from 52 weekly data sets:



```
DATA path_list_files;
  LENGTH fpath $100 fpath_line $1000;
  FORMAT Load_Date date9.;
  INFILE Inpipe TRUNCOVER;
  INPUT fpath $100.;
  Load_Date_Text = SUBSTR(fpath,35,8);
  Load_Date = MDY(INPUT(SUBSTR(Load_Date_Text,5,2),2.),
                  INPUT(SUBSTR(Load_Date_Text,7,2),2.),
                  INPUT(SUBSTR(Load_Date_Text,1,4),4.));
  fpath_line = CATS('DATA file_final_',Load_Date_Text,"; SET '",fpath,
                  "' ; FORMAT Load_Date date9.; Load_Date =",PUT(Load_Date,date9.),
                  "'d; KEEP Special_Person Special_Number Load_Date; RUN;");
RUN;
```

- ❖ The previous section focused on the gray highlighted code. We will now focus on the rest of the code.

#### The remainder of the DATA step will use the following new variables:

```
LENGTH fpath_line $1000;
FORMAT Load_Date date9.;
```

- ❖ The **LENGTH** statement assigns a length of **1000** characters to the new fpath\_line variable which will contain the Dynamic Code of a complete DATA step to create the new data sets.
- ❖ The **FORMAT** statement assigns a format of **date9** (ddmonyyyy) to the new **Load\_Date** variable.

## The new Load\_Date variable is derived from the name of the data set:

```
Load_Date_Text = SUBSTR(fpath,35,8);  
Load_Date = MDY(INPUT(SUBSTR(Load_Date_Text,5,2),2.),  
                INPUT(SUBSTR(Load_Date_Text,7,2),2.),  
                INPUT(SUBSTR(Load_Date_Text,1,4),4.));
```

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

```
/data/MWSUG/PIPE_CALL_EXECUTE/file20130104.sas7bdat  (fpath contents – 1st observation)  
123456789012345678901234567890123456789012345678901  (character spacing)
```

- ❖ The **SUBSTR** function sets **Load\_Date\_Text** to '20130104' – begins with character 35 of **fpath** for 8 characters.

- ❖ The **SUBSTR** function obtains the month '01', day '04', and year '2013' from **Load\_Date\_Text**:

```
Load_Date = MDY(INPUT(SUBSTR('20130104',5,2),2.),  
                INPUT(SUBSTR('20130104',7,2),2.),  
                INPUT(SUBSTR('20130104',1,4),4.));
```

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



```
Load_Date = MDY(INPUT('01',2.),INPUT('04',2.),INPUT('2013',4.));
```

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

```
Load_Date = MDY(1,4,2013);
```

- ❖ Since **Load\_Date** was formatted as **date9** by the earlier **FORMAT** statement, this resolves to:

```
Load_Date = '04JAN2013'd;
```

 Load_Date_Text	 Load_Date
20130104	04JAN2013
20130111	11JAN2013
20130118	18JAN2013
20130125	25JAN2013
20130201	01FEB2013
20130208	08FEB2013
20130215	15FEB2013

**Once Load\_Date is assigned, Load\_Date\_Text, fpath, and Load\_Date  
are used to create the 1st set of Dynamic Code:**

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

- ❖ The **PUT** function is used to convert the **Load\_Date** from a numeric SAS date to a character representation.

- ❖ The **Load\_Date\_Text**, **fpath**, and **Load\_Date** variables resolve to:

```
fpath_line = CATS('DATA file_final_',20130104,
                "; SET '","/data/MWSUG/PIPE_CALL_EXECUTE/file20130104.sas7bdat,
                "','; FORMAT Load_Date date9.; Load_Date =", '04JAN2013'd; ",
                ' KEEP Special_Person Special_Number Load_Date; RUN;');
```

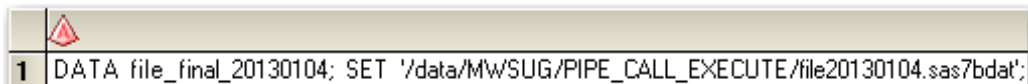
- ❖ The **CATS** function concatenates what is separated by commas while removing leading and trailing spaces:

```
fpath_line = "DATA file_final_20130104;
              SET '/data/MWSUG/PIPE_CALL_EXECUTE/file20130104.sas7bdat';
              FORMAT Load_Date date9.; Load_Date = '04JAN2013'd;
              KEEP Special_Person Special_Number Load_Date;
              RUN;";
```

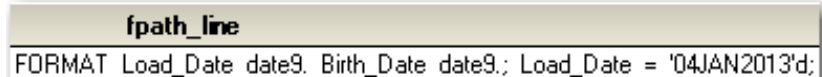
- ❖ You may be asking yourself, "Why do the **FORMAT** statement and the **Load\_Date** assignment appear here since they were already included in the code discussed earlier?"

- ❖ Good question – remember, this Dynamic Code will run apart from the Static Code, so the Dynamic Code needs to be self-contained with all of the statements and syntax necessary to run on its own.

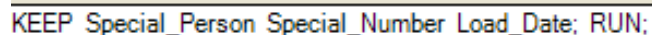
- ❖ The **KEEP** statement enables you to create the output data set with only the vital variables listed:



```
1 DATA file_final_20130104; SET '/data/MWSUG/PIPE_CALL_EXECUTE/file20130104.sas7bdat';
```



```
fpath_line
FORMAT Load_Date date9. Birth_Date date9.; Load_Date = '04JAN2013'd;
```



```
KEEP Special_Person Special_Number Load_Date; RUN;
```

## Once the Dynamic Code is created, then the observation is written:

```
DATA path_list_files;
  LENGTH fpath $100 fpath_date $100 fpath_line $1000;
  FORMAT Load_Date date9.;
  INFILE Inpipe TRUNCOVER;
  INPUT fpath $100.;
  Load_Date_Text = SUBSTR(fpath,35,8);
  Load_Date = MDY(INPUT(SUBSTR(Load_Date_Text,5,2),2.),
                  INPUT(SUBSTR(Load_Date_Text,7,2),2.),
                  INPUT(SUBSTR(Load_Date_Text,1,4),4.));
  fpath_line = CATS('DATA file_final_',Load_Date_Text,"; SET '",fpath,
                  "'; FORMAT Load_Date date9.; Load_Date =",PUT(Load_Date,date9.),
                  "'d; KEEP Special_Person Special_Number Load_Date; RUN;");
RUN;
```

- ❖ The **RUN** statement writes an observation and sends the program to the top to read the next **fpath**.
- ❖ Here is a partial view of the first 2 observations in the **path\_list\_files** data set:

	fpath	fpath_line	Load_Date	Load_Date_Text
1	/data/MWSUG/PIPE_CALL_EXEC...	DATA file_final_20130104; SET '/data/MWSUG/PIPE_CALL_EXECUTE/file20130104.sas7bdat'; FORMAT ...	04JAN2013	20130104
2	/data/MWSUG/PIPE_CALL_EXEC...	DATA file_final_20130111; SET '/data/MWSUG/PIPE_CALL_EXECUTE/file20130111.sas7bdat'; FORMAT ...	11JAN2013	20130111

The first part of **THE POWER TO CREATE** section has walked us through the process of creating Dynamic Code to automatically **Extract** vital variables from 52 weekly data sets and combine them with a Load\_Date variable (created from the Friday date value derived from the Filenames of the data sets) to create 52 new data sets. The **Extract Dynamic Code** is contained in the **fpath\_line** variable.

## Here are the 3 programs displayed as Base SAS Program Nodes:



## Here is 1 observation of the Dynamic Code created by the 3 programs:

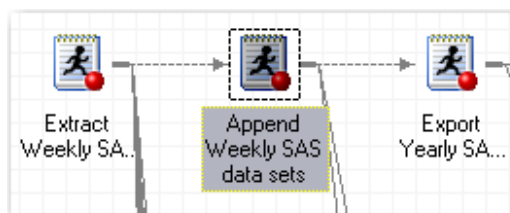
SAS Dataset: PATH\_FILE\_LIST  
Variable: fpath\_line  
DATA file\_final\_20130104;  
SET '/data/MWSUG/  
PIPE\_CALL\_EXECUTE/  
file20130104.sas7bdat';  
FORMAT Load\_Date date9.  
Birth\_Date date9.;  
Load\_Date = '04JAN2013'd;  
KEEP Special\_Person  
Special\_Number  
Load\_Date;  
RUN;

To  
Be  
Illuminated

To  
Be  
Illuminated

The following examples highlight how to create Static Code which automatically creates Dynamic Code to automatically **Append** the 52 new data sets to create a yearly data set.

### How to Append the 52 new data sets to create a yearly data set:



```

DATA prepare_historical_append;
  SET path_list_files END=LAST_OBS;
  LENGTH history_append_line $2000;
  KEEP history_append_line;
  RETAIN history_append_line;
  IF _N_ = 1
    THEN history_append_line = CATS('DATA file_final_',
                                     SUBSTR(Load_Date_Text,1,4),'; SET ');

  history_append_line = SUBSTR(history_append_line,1,
                              LENGTH(history_append_line))||
    CAT(' file_final_',Load_Date_Text,' ');

  IF LAST_OBS THEN
    DO;
      history_append_line = SUBSTR(history_append_line,1,
                                  LENGTH(history_append_line))||'; RUN;';
      OUTPUT;
    END;
  RUN;
  
```

### Creating a DATA step that creates Dynamic Code to Append the data sets:

```

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

- ❖ The **DATA** statement creates an output data set called **prepare\_historical\_append**.
- ❖ The **SET** statement sets **path\_list\_files** as the input data set for this **DATA** step.
- ❖ The **END=LAST\_OBS** option sets **LAST\_OBS** to True as the last observation in **path\_list\_files** is read.
- ❖ The **LENGTH** statement assigns a length of 2000 characters to the **history\_append\_line** variable.
- ❖ The **KEEP** statement creates the output data set with only the **history\_append\_line** variable.
- ❖ The **RETAIN** statement retains the value of **history\_append\_line** throughout the **DATA** step.

## The history\_append\_line variable is derived from the Load\_Date\_Text variable when processing the first observation:

```
IF _N_ = 1
  THEN history_append_line = CATS('DATA file_final_',
                                SUBSTR(Load_Date_Text,1,4),'; SET ');
```

- ❖ The **IF-THEN** statement only executes while processing the first input observation ( $\_N_ = 1$ ).
- ❖ The **Load\_Date\_Text** variable resolves to:  

```
history_append_line = CATS('DATA file_final_',SUBSTR('20130104',1,4),'; SET ');
```
- ❖ The **SUBSTR** function resolves to '2013':  

```
history_append_line = CATS('DATA file_final_', '2013', '; SET ');
```
- ❖ The **CATS** function resolves to:  

```
history_append_line = 'DATA file_final_2013; SET';
```
- ❖ Notice how **history\_append\_line** looks like the beginning of a **DATA** step.

## The history\_append\_line variable is then derived from itself and Load\_Date\_Text again for all observations:

```
history_append_line = SUBSTR(history_append_line,1,
                             LENGTH(history_append_line))||
CAT(' file_final_',Load_Date_Text,' ');
```

- ❖ The **history\_append\_line** and **Load\_Date\_Text** variables resolve to:  

```
history_append_line = SUBSTR('DATA file_final_2013; SET',1,
                             LENGTH('DATA file_final_2013; SET'))||
CAT(' file_final_', '20130104', ' ');
```
- ❖ The **LENGTH** and **CAT** (concatenates but keeps spaces) functions resolve to:  

```
history_append_line = SUBSTR('DATA file_final_2013; SET',1,25)||
' file_final_20130104 ';
```
- ❖ The **SUBSTR** function resolves to the way **history\_append\_line** looked at the end of the previous assignment statement:  

```
history_append_line = 'DATA file_final_2013; SET' || ' file_final_20130104 ';
```
- ❖ The **||** concatenates whatever is on both sides of it while keeping the formatting intact:  

```
history_append_line = 'DATA file_final_2013; SET file_final_20130104 ';
```



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

```
history_append_line = SUBSTR(history_append_line,1,
                             LENGTH(history_append_line))||
                             CAT(' file_final_',Load_Date_Text,' ');
```

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

```
history_append_line = 'DATA file_final_2013; SET file_final_20130104
                    file_final_20130111
                    ... ';
```

- ❖ This will continue until the last Filename is added:

```
history_append_line = 'DATA file_final_2013; SET file_final_20130104
                    ...
                    file_final_20130215';
```

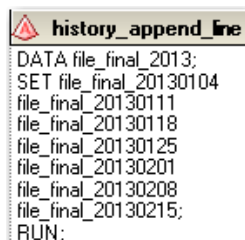
## The `history_append_line` variable is then derived from itself and `Load_Date_Text` a final time for the last observation:

```
IF LAST_OBS THEN
  DO;
    history_append_line = SUBSTR(history_append_line,1,
                                LENGTH(history_append_line))||'; RUN;';
    OUTPUT;
  END;
```

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

```
history_append_line = 'DATA file_final_2013; SET file_final_20130104
                    ...
                    file_final_20130215; RUN;';
```

- ❖ The `OUTPUT` statement is executed within `IF-THEN DO-END` because only one observation is needed in the output data set containing the completed `history_append_line`.
- ❖ Here is how the **only** observation appears in the `prepare_historical_append` data set:

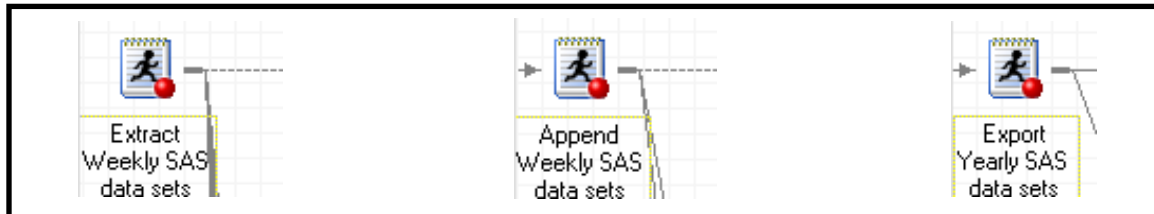


The screenshot shows a SAS data set viewer window titled 'history\_append\_line'. It displays a single observation with the following text:

```
DATA file_final_2013;
SET file_final_20130104
file_final_20130111
file_final_20130118
file_final_20130125
file_final_20130201
file_final_20130208
file_final_20130215;
RUN;
```

The second part of **THE POWER TO CREATE** section has walked us through the process of creating Dynamic Code to automatically **Append** the 52 new data sets to create a yearly data set. The **Append Dynamic Code** is contained in the **history\_append\_line** variable.

**Here are the 3 programs displayed as Base SAS Program Nodes:**



**Here is 1 observation of the Dynamic Code created by the 3 programs:**

SAS Dataset: PATH\_FILE\_LIST  
Variable: fpath\_line

```
DATA file_final_20130104;
SET '/data/MWSUG/
    PIPE_CALL_EXECUTE/
    file20130104.sas7bdat';
FORMAT Load_Date date9.
    Birth_Date date9.;
Load_Date = '04JAN2013'd;
KEEP Special_Person
    Special_Number
    Load_Date;
RUN;
```

SAS Dataset:  
prepare\_historical\_append  
Variable: history\_append\_line

```
DATA file_final_2013;
SET file_final_20130104
    file_final_20130111
    file_final_20130118
    file_final_20130125
    file_final_20130201
    file_final_20130208
    file_final_20130215;
RUN;
```

To  
Be  
Illuminated

The following examples highlight how to create Static Code which automatically creates Dynamic Code to automatically **Export** the yearly data set.

### How to Export the appended yearly data set back to the server:



```
DATA LIST_OF_SAS_DATASETS_TO_EXPORT;  
  SET path_list_files;  
  KEEP export_line;  
  IF _N_ = 1;  
    export_line = CATS("DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_",  
                      SUBSTR(Load_Date_Text,1,4));  
    export_line = SUBSTR(export_line,1,LENGTH(export_line))||  
                  CATS("; SET file_final_");  
    export_line = SUBSTR(export_line,1,LENGTH(export_line))||  
                  CATS(SUBSTR(Load_Date_Text,1,4),'; RUN;');  
RUN;
```

### Creating a DATA step that creates Dynamic Code to Export a data set:

```
DATA LIST_OF_SAS_DATASETS_TO_EXPORT;  
  SET path_list_files;  
  KEEP export_line;  
  IF _N_ = 1;
```

- ❖ The **DATA** statement creates an output data set called **LIST\_OF\_SAS\_DATASETS\_TO\_EXPORT**.
- ❖ The **SET** statement sets **path\_list\_files** as the input data set for this **DATA** step.
- ❖ The **KEEP** statement creates the output data set with only the **export\_line** variable.
- ❖ The **IF \_N\_ = 1** statement executes the rest of the DATA step while only processing the first input observation (**\_N\_ = 1**).

## Load\_Date\_Text is used to create Dynamic Code:

```
export_line = CATS("DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_",
                  SUBSTR(Load_Date_Text,1,4));
export_line = SUBSTR(export_line,1,LENGTH(export_line))||
              CATS("; SET file_final_");
export_line = SUBSTR(export_line,1,LENGTH(export_line))||
              CATS(SUBSTR(Load_Date_Text,1,4),'; RUN;');
RUN;
```

- ❖ The first **export\_line** with the **Load\_Date\_Text** variable resolves to:

```
export_line = CATS("DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_",
                  SUBSTR('20130104',1,4));
```

- ❖ The **SUBSTR** function resolves to '2013':

```
export_line = CATS("DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_", '2013');
```

- ❖ The **CATS** function resolves to:

```
export_line = 'DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_2013';
```

- ❖ The second **export\_line** resolves to the way it looked at the end of the previous assignment statement and then concatenates with the result of the **CATS** function:

```
export_line = "DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_2013"||
              CATS("; SET file_final_");
```

- ❖ The **CATS** function resolves to and the result concatenates to:

```
export_line="DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_2013'; SET file_final_";
```

- ❖ The third **export\_line** resolves to the way it looked at the end of the previous assignment statement and then concatenates with the result of the **CATS** function:

```
export_line = "DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_2013';
              SET file_final_"||CATS(SUBSTR('20130104',1,4),'; RUN;');
```

- ❖ The **SUBSTR** resolves to '2013':

```
export_line = "DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_2013';
              SET file_final_"||CATS('2013','; RUN;');
```

- ❖ The **CATS** function resolves to '2013; RUN;':

```
export_line = "DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_2013';
              SET file_final_"||'2013; RUN;';
```

- ❖ The final **export\_line** resolves to:

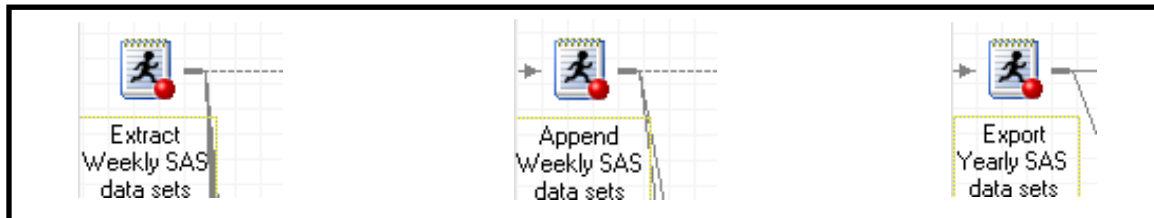
```
export_line = "DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_2013';
              SET file_final_2013;
              RUN;";
```

- ❖ Here is how the **only** observation appears in the **LIST\_OF\_SAS\_DATASETS\_TO\_EXPORT** data set:

	export_line
1	DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_2013'; SET file_final_2013; RUN;

The third part of **THE POWER TO CREATE** section has walked us through the process of creating Dynamic Code to automatically **Export** a yearly data set back to the folder on the server where the weekly snapshot data sets are stored. The **Export Dynamic Code** is contained in the **export\_line** variable.

### Here are the 3 programs displayed as Base SAS Program Nodes:



### Here is 1 observation of the Dynamic Code created by the 3 programs:

SAS Dataset: PATH\_FILE\_LIST  
Variable: fpath\_line

```
DATA file_final_20130104;
  SET '/data/MWSUG/
    PIPE_CALL_EXECUTE/
    file20130104.sas7bdat';
  FORMAT Load_Date date9.
         Birth_Date date9.;
  Load_Date = '04JAN2013'd;
  KEEP Special_Person
        Special_Number
        Load_Date;
RUN;
```

SAS Dataset:  
prepare\_historical\_append  
Variable: history\_append\_line

```
DATA file_final_2013;
  SET file_final_20130104
      file_final_20130111
      file_final_20130118
      file_final_20130125
      file_final_20130201
      file_final_20130208
      file_final_20130215;
RUN;
```

SAS Dataset:  
LIST\_OF\_SAS\_DATASETS\_  
TO\_EXPORT  
Variable: export\_line

```
DATA '/data/MWSUG/
  PIPE_CALL_EXECUTE/
  file_all_2013';
  SET file_final_2013;
RUN;
```

## THE POWER TO EXECUTE

### Dynamic Code Automatically Using The CALL EXECUTE Command

After the Dynamic Code has been created the CALL EXECUTE command is used to execute the 3 sets of Dynamic Code automatically to **Extract, Append, and Export** the appended yearly data set.

#### Executing the Extract Dynamic Code using the CALL EXECUTE command:

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

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

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

- ❖ The **DATA** statement does not create an output data set because the **\_NULL\_** option is used.
- ❖ The **SET** statement sets **path\_list\_files** as the input data set for this **DATA** step.
- ❖ Here is a partial view of the first 2 observations in the **path\_list\_files** data set:

	fpath	fpath_line	Load_Date	Load_Date_Text
1	/data/MWSUG/PIPE_CALL_EXEC...	DATA file_final_20130104; SET /data/MWSUG/PIPE_CALL_EXECUTE/file20130104.sas7bdat; FORMAT...	04JAN2013	20130104
2	/data/MWSUG/PIPE_CALL_EXEC...	DATA file_final_20130111; SET /data/MWSUG/PIPE_CALL_EXECUTE/file20130111.sas7bdat; FORMAT...	11JAN2013	20130111

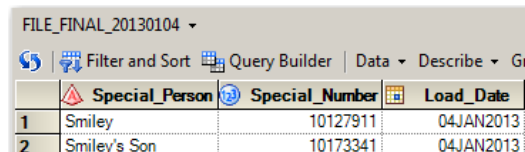
## The CALL EXECUTE command executes the fpath\_line variable:

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

- ❖ The **CALL EXECUTE** command executes the contents of the **fpath\_line** variable in the **path\_list\_files** data set. Here is the first observation of **fpath\_line** in the **path\_list\_files** data set:

```
DATA file_final_20130104;  
SET '/data/MWSUG/PIPE_CALL_EXECUTE/file20130104.sas7bdat';  
FORMAT Load_Date date9. Birth_Date date9.;  
Load_Date = '04JAN2013'd;  
KEEP Special_Person Special_Number Load_Date;  
RUN;
```

- ❖ Here is the result of executing the first observation of **fpath\_line** in the **path\_list\_files** data set:

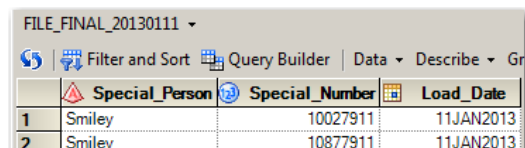


	Special_Person	Special_Number	Load_Date
1	Smiley	10127911	04JAN2013
2	Smiley's Son	10173341	04JAN2013

- ❖ The **RUN** statement causes the second observation of **fpath\_line** in the **path\_list\_files** data set to be read:

```
DATA file_final_20130111;  
SET '/data/MWSUG/PIPE_CALL_EXECUTE/file20130111.sas7bdat';  
FORMAT Load_Date date9. Birth_Date date9.;  
Load_Date = '11JAN2013'd;  
KEEP Special_Person Special_Number Load_Date;  
RUN;
```

- ❖ Here is the result of executing the second observation of **fpath\_line** in the **path\_list\_files** data set:



	Special_Person	Special_Number	Load_Date
1	Smiley	10027911	11JAN2013
2	Smiley	10877911	11JAN2013

- ❖ The execution of **fpath\_line** continues for each observation in the **path\_list\_files** data set.

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

## Executing the Append Dynamic Code using the CALL EXECUTE command:

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



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

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

- ❖ The **DATA** statement does not create an output data set because the **\_NULL\_** option is used.
- ❖ The **SET** statement sets **prepare\_historical\_append** as the input data set for this **DATA** step.
- ❖ Here is the **only** observation in the **prepare\_historical\_append** data set:

history_append_line	
1	DATA file_final_2013; SET file_final_20130104 file_final_20130111 file_final_20130118 file_final_20130125 file_final_20130201 file_final_20130208; RUN;

## The CALL EXECUTE command executes the history\_append\_line variable:

```
CALL EXECUTE(history_append_line);  
RUN;
```

- ❖ The **CALL EXECUTE** command executes the contents of the **history\_append\_line** variable in the **prepare\_historical\_append** data set. Here is the **only** observation of **history\_append\_line**:

```
DATA file_final_2013;  
    SET file_final_20130104  
        file_final_20130111  
        file_final_20130118  
        file_final_20130125  
        file_final_20130201  
        file_final_20130208  
        file_final_20130215;  
  
RUN;
```

- ❖ Here is the result of executing **history\_append\_line** in the **prepare\_historical\_append** data set:

FILE_FINAL_2013		
	Special_Person	Special_Number
1	Smiley	10127911
2	Smiley's Son	10173341

Now that the Dynamic Code has been executed to automatically **Append** the **52** new data sets, the final step is to execute the **Export Dynamic Code**.

## Executing the Export Dynamic Code using the CALL EXECUTE command:

```
DATA _NULL_;  
    SET LIST_OF_SAS_DATASETS_TO_EXPORT;  
    CALL EXECUTE(export_line);  
RUN;
```

## Creating a DATA step that executes Dynamic Code to Export data sets:

```
DATA _NULL_ ;  
    SET LIST_OF_SAS_DATASETS_TO_EXPORT ;
```

- ❖ The **DATA** statement does not create an output data set because the **\_NULL\_** option is used.
- ❖ The **SET** statement sets **LIST\_OF\_SAS\_DATASETS\_TO\_EXPORT** as the input data set for this **DATA** step.
- ❖ Here is the **only** observation in the **LIST\_OF\_SAS\_DATASETS\_TO\_EXPORT** data set:



export_line
DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_2013'; SET file_final_2013; RUN;

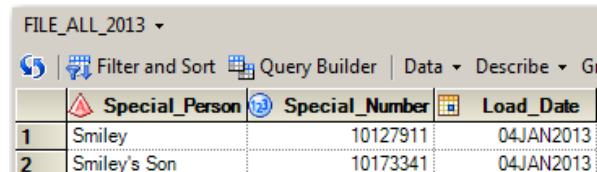
## The CALL EXECUTE command executes the fpath\_line variable:

```
CALL EXECUTE (export_line) ;  
RUN;
```

- ❖ The **CALL EXECUTE** command executes the contents of the **export\_line** variable in the **LIST\_OF\_SAS\_DATASETS\_TO\_EXPORT** data set. Here is the **only** observation of **export\_line**:

```
DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_2013';  
SET file_final_2013;  
RUN;
```

- ❖ Here is the result of executing **export\_line** in the **LIST\_OF\_SAS\_DATASETS\_TO\_EXPORT** data set:



	Special_Person	Special_Number	Load_Date
1	Smiley	10127911	04JAN2013
2	Smiley's Son	10173341	04JAN2013

Now that we have completed the process for **1** year, we need to repeat the process for the remaining **10** years for this project. How is this accomplished? We simply **update the year** in the PIPE command portion of the FILENAME statement in the **Extract Program Node**, rerun all **3** Program Nodes, and then repeat this process until each of the remaining years is complete.

## Creating the Yearly data sets for each Year:

```
FILENAME Inpipe PIPE 'ls /data/MWSUG/PIPE_CALL_EXECUTE/file2013*.sas7bdat';
```

- ❖ Update the Year in the Extract Program Node and then rerun all three Program Nodes for each year.

☺ Done and Done ☺

## CONCLUSION

**The Power To Know** through the **PIPE** command enables **The Power To Create** Static Code which automatically creates Dynamic Code and leads to **The Power To Execute** the Dynamic Code automatically using the **CALL EXECUTE** command. (Try saying that statement really fast for fun 😊.) Our presentation has shown you how **1,259** time-consuming Manual Steps are amazingly replaced with only **3** time-saving Dynamic Automated Steps.

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



*It's not what the world holds for you,  
it's what YOU bring to it!*

Anne of Green Gables



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

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

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



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

Claire Draper

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

As we conclude our presentation, we want to introduce you to our **SAS Mascot, Smiley**. Smiley represents the **SAS Joy** which each of us experience when we find better ways to accomplish mighty and worthy deeds using SAS. The four of us hope that the time we have shared together has expanded and enriched your SAS knowledge. You may or may not use the PIPE and CALL EXECUTE commands 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 Honoring Us With Your Participation**

😊 **Happy SAS Trails To You... Until We Meet Again** 😊



## MEET THE AUTHORS

***Writing is a permanent legacy.***

**John C. Maxwell**

**Kent Phelps** (*Senior Data Governance Analyst*) has worked in IT and Data Governance since 1990, has programmed in SAS since 2007, is a SAS Certified Professional specializing in combining and automating the best of SAS Enterprise Guide with Base SAS, has Co-Created/Led *Intro To SAS EG* classes, and presents *SAS News You Can Use*. Kent has a B.S. in Electrical Engineering, has studied Transformational Leadership, Dynamic Teamwork, and Personal Growth since 1994, and is certified as a *John Maxwell Team* coach and a *48 Days To The Work You Love* coach. His hope is to encourage you to fulfill your life and leadership potential and to equip you in building an enduring legacy of inspiration, excellence, and honor.

\*\*\*\*\*

**Ronda Phelps** (*Writer, Teacher, and Coach*) formerly worked in the Banking and Insurance industries for 19 years, has studied Transformational Leadership, Dynamic Teamwork, and Personal Growth since 1994, and is certified as a *John Maxwell Team* coach and a *48 Days To The Work You Love* coach. She believes YOU are a gift that the world is waiting to receive! Her hope is to encourage you to pursue your unique destiny and to equip you in navigating your journey with intentionality, fulfilling purpose, and enduring hope.

\*\*\*\*\*

**Kirk Paul Lafler** (*Founder/Senior Consultant, Software Intelligence Corporation*) has programmed in SAS since 1979, is a SAS Certified Professional, provides IT Consulting Services, trains and mentors SAS users worldwide, and is a SAScommunity.org Emeritus Advisory Board member. Kirk has authored 6 books including *Google Search Complete!* (Odyssey Press 2014), has written over 500 papers and articles, has been invited to speak and/or train at over 400 SAS international, regional, special-interest, local and in-house user group conferences and meetings, and has received 23 BEST Contributed Paper, Hands-On Workshop (HOW), and Poster Awards. His popular SAS Tips column *Kirk's Korner of Quick and Simple Tips* and his funfilled *SASword Puzzles* appear on various SAS websites and in several SAS User Group newsletters.

**We invite you to share your valued comments with us:**

**Kent ♥ Ronda Team Phelps**

Writers, Teachers, and Coaches

E-mail: [SASketeers@q.com](mailto:SASketeers@q.com)

**Kirk Paul Lafler**

Senior Consultant, Application Developer, Trainer, Mentor, and Author

Software Intelligence Corporation

E-mail: [KirkLafler@cs.com](mailto:KirkLafler@cs.com)

LinkedIn: <http://www.linkedin.com/in/KirkPaulLafler>

Twitter: @sasNerd

☺ **We Look Forward To Connecting With You In The Future** ☺

## APPENDIX A

### PIPE and CALL EXECUTE and Dynamic Code Syntax for Windows and z/OS

**Disclaimer:** 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.

Our project example details the **UNIX** syntax for the **PIPE** and **CALL EXECUTE** commands and the **Dynamic Code**. This Appendix is a starting point regarding the syntax for **Windows** and **z/OS**.

#### Creating the FILENAME statement on page 7:

```
FILENAME Inpipe PIPE 'ls /data/MWSUG/PIPE_CALL_EXECUTE/file2013*.sas7bdat';
```

- ❖ The **Windows** version of the **FILENAME** statement uses the **dir** OS command to create the Directory Listing while also referencing the specific drive letter and the record length of the **PIPE** result:

```
FILENAME Inpipe PIPE 'dir "c:\data\MWSUG\PIPE_CALL_EXECUTE\file2013*.sas7bdat" /S'
lrecl=100;
```

- ❖ The **z/OS** version of the **FILENAME** statement can take different forms depending on the **z/OS** version and installation configuration. Here are 2 reference links as a starting point:

- **Allocating External Files to a Pipe through BatchPIPEs from SAS® 9.3 Companion for z/OS:**  
<http://support.sas.com/documentation/cdl/en/hosto390/65144/HTML/default/viewer.htm#n0fxdtzgeaxa51n1080fzebl81qx.htm>
- **Accessing UNIX System Services Files from SAS® 9.3 Companion for z/OS:**  
<http://support.sas.com/documentation/cdl/en/hosto390/65144/HTML/default/viewer.htm#n001udyg5mzcb1n1bh8ts48m1bal1.htm>

#### Creating the first Dynamic Code export\_line on page 18:

```
export_line = CATS("DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file_all_",
```

- ❖ The **Windows** version of the **export\_line** statement uses the specific drive letter:  

```
export_line = CATS("DATA 'c:\data\MWSUG\PIPE_CALL_EXECUTE\file_all_",
```
- ❖ The **z/OS** version of the **export\_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® 9.3 Companion for z/OS:**  
<http://support.sas.com/documentation/cdl/en/hosto390/65144/HTML/default/viewer.htm#p1t2wsrhr9x099n1h967cql2j3fm.htm>
- **SAS® 9.3 Companion for z/OS:**  
<http://support.sas.com/documentation/cdl/en/hosto390/65144/HTML/default/viewer.htm#titlepage.htm>

## Executing the first CALL EXECUTE command on page 21:

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

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

## APPENDIX B

### Alternative for the PIPE Command for Our Project Example

The **Power To Know** section demonstrated how to use the PIPE command to request and receive one Directory Listing of the Filenames of the 52 weekly data sets for each year from a folder on the server. The following example demonstrates how to achieve the same results using an alternative for the PIPE command.

#### How to request and receive a Directory Listing without the PIPE command:

```
FILENAME Nopipe '/data/MWSUG/PIPE_CALL_EXECUTE/file2013*.sas7bdat';

DATA path_list_files;
  LENGTH SAS_path_and_data_set fpath $100 fpath_line $1000;
  FORMAT Load_Date date9.;
  DO UNTIL (DONE);
    INFILE Nopipe TRUNCOVER FILENAME=SAS_path_and_data_set END=DONE;
    INPUT;
    IF fpath NE SAS_path_and_data_set THEN
      DO;
        fpath = SAS_path_and_data_set;
        Load_Date_Text = SUBSTR(fpath,65,8);
        Load_Date = MDY(INPUT(SUBSTR(Load_Date_Text,5,2),2.),
                        INPUT(SUBSTR(Load_Date_Text,7,2),2.),
                        INPUT(SUBSTR(Load_Date_Text,1,4),4.));
        fpath_line = CATS('DATA file_final_',Load_Date_Text,"; SET '",fpath,
                        "'; FORMAT Load_Date date9.; Load_Date = '",PUT(Load_Date,date9.),
                        "'d; KEEP Special_Person Special_Number Load_Date; RUN;");
        OUTPUT;
      END;
    END;
  END;
RUN;
```

- ❖ Previous sections focused on the `gray` highlighted code. We will now focus on the rest of the code.

#### Creating a FILENAME statement without the PIPE command:

```
FILENAME Nopipe '/data/MWSUG/PIPE_CALL_EXECUTE/file2013*.sas7bdat';
```

- ❖ The **FILENAME** statement assigns **Nopipe** as a file reference (fileref) to the folder and file pattern.

#### Creating a variable which will store and track the Directory Listing:

```
LENGTH SAS_path_and_data_set fpath $100 fpath_line $1000;
```

- ❖ The **LENGTH** statement assigns a length of **100** characters to a variable that will be used to store and track changes to the data set path and name called **SAS\_path\_and\_data\_set**.



## Creating a DO UNTIL loop to process each Filename in the Directory Listing:

```
DO UNTIL (DONE) ;  
  INFILE Nopipe TRUNCOVER FILENAME=SAS_path_and_data_set END=DONE;  
  INPUT;  
  ...  
END;
```

- ❖ The **DO UNTIL (DONE)** statement executes the loop through the **END** statement until **DONE** is true.
- ❖ The **INFILE** statement assigns **Nopipe** (fileref) to the folder and file pattern to be read with the upcoming **INPUT** statement.
- ❖ The **TRUNCOVER** option tells SAS the input data may or may not be the same length.
- ❖ The **END=DONE** sets **DONE** to true when the last record in the last file fitting the file pattern is read.
- ❖ The **FILENAME=SAS\_path\_and\_data\_set** statement assigns **SAS\_path\_and\_data\_set** to the path and name of the file being read.
- ❖ The **INPUT** statement reads the **INFILE Nopipe** (fileref) one record at a time.
- ❖ The **END** statement sends control to the top of the **DO UNTIL** loop so the next record can be read.

## Creating an IF-THEN DO-END statement to detect new Filenames being read:

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

- ❖ The **IF fpath NE SAS\_path\_and\_data\_set THEN** statement executes the contents of the **DO-END** when a new file is read.
- ❖ The **fpath = SAS\_path\_and\_data\_set** statement assigns **fpath** to **SAS\_path\_and\_data\_set** which contains the path and name of the file being read.
- ❖ The **fpath** variable is reassigned to each new Filename as the Filename changes.
- ❖ The **fpath** variable is then used to build the Dynamic Code as described in previous sections.
- ❖ The **OUTPUT** statement is executed within the **IF-THEN DO-END** statement to ensure that we only create Dynamic Code and write an observation when **fpath** changes.

Combining the PIPE alternative with the gray highlighted code (shown on page 29) enables the same Directory Listing and Dynamic Code to be created as we did using the PIPE command. As a result, no additional changes are required for Program Nodes 2 and 3.

## APPENDIX C

### The Code that Created the Data Sets for Our Project Example

```
DATA '/data/MWSUG/PIPE_CALL_EXECUTE/file20130104.sas7bdat'
      '/data/MWSUG/PIPE_CALL_EXECUTE/file20130111.sas7bdat'
      '/data/MWSUG/PIPE_CALL_EXECUTE/file20130118.sas7bdat'
      '/data/MWSUG/PIPE_CALL_EXECUTE/file20130125.sas7bdat'
      '/data/MWSUG/PIPE_CALL_EXECUTE/file20130201.sas7bdat'
      '/data/MWSUG/PIPE_CALL_EXECUTE/file20130208.sas7bdat'
      '/data/MWSUG/PIPE_CALL_EXECUTE/file20130215.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/PIPE_CALL_EXECUTE/file20130104.sas7bdat';
      WHEN (N_ LE 10)  OUTPUT '/data/MWSUG/PIPE_CALL_EXECUTE/file20130111.sas7bdat';
      WHEN (N_ LE 15)  OUTPUT '/data/MWSUG/PIPE_CALL_EXECUTE/file20130118.sas7bdat';
      WHEN (N_ LE 20)  OUTPUT '/data/MWSUG/PIPE_CALL_EXECUTE/file20130125.sas7bdat';
      WHEN (N_ LE 25)  OUTPUT '/data/MWSUG/PIPE_CALL_EXECUTE/file20130201.sas7bdat';
      WHEN (N_ LE 30)  OUTPUT '/data/MWSUG/PIPE_CALL_EXECUTE/file20130208.sas7bdat';
      OTHERWISE        OUTPUT '/data/MWSUG/PIPE_CALL_EXECUTE/file20130215.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 **Dave Foster**, the 25<sup>th</sup> Annual MWSUG 2014 BI/CI Section Chair, and **Bruce Lund**, the Asst. BI/CI Section Chair, for graciously accepting our abstract and paper. In addition, we want to express our appreciation to the Co-Chairs, **Cindy Lee** (Academic Chair) and **Craig Wildeman** (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 each of you who empower us through your teaching endeavors. Your heart to continuously share what you are learning, blended with your servant leadership and supportive guidance, is a constant light of encouragement to us. You inspire us to keep sharing what we are learning and our hope is to be a light of encouragement to you as well – All for One & One 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 23<sup>rd</sup> Annual MidWest SAS Users Group (MWSUG) 2012 Conference, Software Intelligence Corporation, Spring Valley, CA, USA, and Consider Consulting, Inc., San Pedro, CA, USA.  
<http://www.mwsug.org/proceedings/2012/JM/MWSUG-2012-JM06.pdf>

**Lafler, Kirk Paul (2012)**, *You Could Be a SAS® Nerd If. . .*, Proceedings of the 23<sup>rd</sup> Annual MidWest SAS Users Group (MWSUG) 2012 Conference, Software Intelligence Corporation, Spring Valley, CA, USA.  
<http://www.mwsug.org/proceedings/2012/S1/MWSUG-2012-S103.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 30<sup>th</sup> 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 Phelps and Kirk Paul Lafler (2014)**, *The Joinless Join; Expand the Power of SAS® Enterprise Guide® in a New Way*, Proceedings of the 25<sup>th</sup> Annual MidWest SAS Users Group (MWSUG) 2014 Conference, The SASketeers, Des Moines, IA, USA, and Software Intelligence Corporation, Spring Valley, CA, USA.

**Phelps, Kent ♥ Ronda Team Phelps 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, USA, and Software Intelligence Corporation, Spring Valley, CA, USA.

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

**Phelps, Kent ♥ Ronda Team Phelps 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, USA, and Software Intelligence Corporation, Spring Valley, CA, USA.

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

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

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

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

**Support.SAS.com (2008)**, *Using External Files and Devices: Reading from and Writing to UNIX Commands (PIPE)*. <http://support.sas.com/documentation/cdl/en/hostunix/61879/HTML/default/viewer.htm#pipe.htm>

**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.