

Leverage Your Programming with Code Generation

Paulette Staum

February 2010

1

Introduction

What is code generation?

- Using a program to write other programs

Why use code generation?

- Write a program once
 - Use it in different situations
- Make changes to one program
 - Instead of many programs

2

How to generate code

- Macros
- Import and Export Wizards
- Display Manager Query and Report Windows
- Enterprise Guide tasks
- DATA step techniques
 - %INCLUDE
 - CALL EXECUTE

3

Outline

- %INCLUDE
- CALL EXECUTE

4

%INCLUDE Syntax

%INCLUDE *source(s)* ;

- When you execute a %INCLUDE statement, the statements and data lines in the source file are executed.
 - Could be part of a step or could be many steps
 - Source could be a permanent or temporary file

5

%INCLUDE Syntax

%INCLUDE *source(s)* / SOURCE2;

- SOURCE2 vs. NOSOURCE2 controls whether the included lines are displayed in the log.
 - There is also a system option SOURCE2 / NOSOURCE2.

6

%INCLUDE Strategy

1. Create text file containing SAS statements
 - Permanent or temporary
2. %INCLUDE the text file

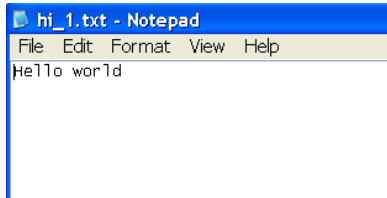
7

Create a text file

```
data _null_;  
  file 'c:\test\hi_1.txt';  
  put "Hello world";  
run;
```

8

Text file output



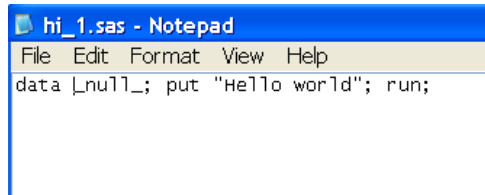
9

Create a text file that is a program

```
data _null_;  
  file 'c:\test\hi_1.sas';  
  put 'data _null_; put "Hello world"; run;' ;  
run;
```

10

Text file output – It's a program!



```
hi_1.sas - Notepad
File Edit Format View Help
data _null_; put "Hello world"; run;
```

11

Executing a program that's in a text file

```
%include 'c:\test\hi_1.sas' / source2;
```

12

```
data _null_;
  file 'c:\test\hi_1.sas';
  put 'data _null_; put "Hello world"; run;';
run;
```

NOTE: The file 'c:\test\hi_1.sas' is:
 Filename=c:\test\hi_1.sas,
 RECFM=V,LRECL=256,File Size (bytes)=0,
 Last Modified=19Feb2010:12:47:47,
 Create Time=19Feb2010:11:18:51

NOTE: 1 record was written to the file 'c:\test\hi_1.sas'.
 The minimum record length was 36.
 The maximum record length was 36.

NOTE: DATA statement used (Total process time):

```
real time    0.00 seconds
cpu time     0.00 seconds
```

```
%include 'c:\test\hi_1.sas'/source2;
```

NOTE: %INCLUDE (level 1) file c:\test\hi_1.sas is file c:\test\hi_1.sas.

```
+data _null_; put "Hello world"; run;
```

Hello world

NOTE: DATA statement used (Total process time):

```
real time    0.00 seconds
cpu time     0.00 seconds
```

NOTE: %INCLUDE (level 1) ending.

13

Logical Reference to a Permanent File

```
filename gencode 'c:\test\hi_1.sas';
```

```
data _null_;
  file gencode;
  put 'data _null_; put "Hello world"; run;';
run;
```

```
%include gencode / source2;
```

```
filename gencode clear;
```

14

Logical Reference to a Temporary File

```
filename gencode temp;  
  
data _null_  
  file gencode;  
  put 'data _null_; put "Hello world"; run;'  
run;  
  
%include gencode / source2;  
  
filename gencode clear;
```

15

Questions?

When would you use a temporary file reference?

16

Example of %INCLUDE

Copy labels from a data dictionary

- Goal: Take variable labels from a data dictionary in an Excel spreadsheet and apply to a data set
- Method:
 - Import labels from Excel spreadsheet to a data set
 - Read data set with labels and create text file with PROC DATASETS statements to add variable labels
 - %INCLUDE text file with statements

Demonstrate generating dynamic code

17

Labels in Excel Spreadsheet

Variable	Label
NAICS	North America Industry Classification System
Description	Industry
TotalSales	Total Sales
PerCapitaSales	Per Capita Sales

18

PROC IMPORT

```
proc import
    datafile= 'c:\test\labels.xls'
    dbms=excel
    out=work.labels replace;
    getnames=yes;
run;
```

Alternative approach if you don't have SAS/ACCESS for PC Files:
 1 - Convert external source to text file
 2 - Use DATA step to read text file

19

Work.Labels

	Variable	Label
1	NAICS	North America Industry Classification System
2	Description	Industry
3	TotalSales	Total Sales
4	PerCapitaSales	Per Capita Sales

20

The Goal: Code to Generate

```
proc datasets lib=state nolist;
  modify industry;
  label NAICS = 'North American Industry
                Classification System';
  label Description = 'Industry';
  label TotalSales = 'Total Sales';
  label PerCapitaSales = 'Per Capita Sales';
quit;
```

Alternative: Use LABEL statements in DATA step

21

Program to Generate Code: Framework

```
filename gencode temp;

data _null_;
  file gencode;
  set work.labels end=eof;
  /* put ... */
run;

%include gencode / source2;
filename gencode clear;
```

22

Program to Generate Code: Logic

```
filename gencode temp;
data _null_;
  file gencode;
  set work.labels end=eof;
  if _n_ = 1 then
    put "proc datasets lib=state  nolist;" /
      "  modify Industry;";
  put "  label "
      variable
      " = "
      label
      " ;";
  if eof then put "quit;";
run;
%include gencode / source2;
filename gencode clear;
```

23

```
filename gencode temp;
data _null_;
  set work.labels end=eof;
  file gencode;
  if _n_ = 1 then
    put "proc datasets lib=state  nolist;" /
      "  modify Industry;";
  put "  label " variable_ " = " label_ " ";";
  if eof then put "quit;";
run;
```

NOTE: The file GENCODE is:

```
Filename=C:\DOCUME~1\Staum\LOCALS~1\Temp\SAS Temporary Files\_TD7864\#LN00025,
RECFM=V,LRECL=256,File Size (bytes)=0,
Last Modified=19Feb2010:12:34:51,
Create Time=19Feb2010:12:34:51
```

NOTE: 7 records were written to the file GENCODE.

The minimum record length was 5.

The maximum record length was 65.

NOTE: There were 4 observations read from the data set WORK.LABELS.

NOTE: DATA statement used (Total process time):

```
real time    0.01 seconds
cpu time     0.01 seconds
```

24

```
%include gencode / source2;
NOTE: %INCLUDE (level 1) file GENCODE is file C:\DOCUME~1\Staum\LOCALS~1\Temp\SAS
Temporary
Files\_TD7864\#LN00025.
+proc datasets lib=state nolist;
+ modify Industry;
+ label NAICS = 'North America Industry Classification System ';
+ label Description = 'Industry ';
+ label TotalSales = 'Total Sales ';
+ label PerCapitaSales = 'Per Capita Sales ';
+quit;

NOTE: MODIFY was successful for STATE.INDUSTRY.DATA.
NOTE: PROCEDURE DATASETS used (Total process time):
    real time    0.01 seconds
    cpu time     0.01 seconds

NOTE: %INCLUDE (level 1) ending.
201 filename gencode clear;
NOTE: Fileref GENCODE has been deassigned.
```

25

Questions?

What would happen if a label contained a single quote?

26

%INCLUDE Summary

- Advantages
 - Can build file in several steps
 - Can save statements for future use
 - Can use an existing file
 - Can contain DATALINES or CARDS statements
 - Can be generated by PROC SQL
- Disadvantages
 - Need to be careful with quotes in text

27

Outline

- %INCLUDE
- **CALL EXECUTE**

28

CALL EXECUTE

```
data _null_;
  ...
  CALL EXECUTE(argument);
  ...
run;
```

- Use in a DATA step
- “Resolves the argument, and issues the resolved value for execution”
- Adds SAS statements to the program stack for execution after the end of the DATA step

29

CALL EXECUTE

The argument can be a character string, variable or a combination of character expressions and variables.

```
data _null_;
  call execute('proc print data=sales;
              run;');
run;
```

```
data _null_;
  findobs='proc print data=sales; run;';
  call execute(findobs);
run;
```

Are these examples realistic?

30

```
data _null_ ; call execute('proc print data=sales; run;'); run;
```

NOTE: DATA statement used (Total process time):

real time	0.00 seconds
cpu time	0.00 seconds

NOTE: CALL EXECUTE generated line.

```
1 + proc print data=sales; run;
```

NOTE: There were 1 observations read from the data set WORK.SALES.

NOTE: PROCEDURE PRINT used (Total process time):

real time	0.12 seconds
cpu time	0.00 seconds

31

Examples of CALL EXECUTE

1. Run steps conditionally
2. Data-based do loop
3. Execute macros
4. Process selected data sets in a library
5. Run data-based, dynamic queries
6. Run all programs in a directory

32

1: Execute steps conditionally

- Goal: Print list of overdue bills, if many bills are overdue
- Method:
 - Check due date for each bill
 - Output observation if bill overdue
 - If many are overdue, print list

1: Demonstrate executing fixed, constant code

33

1: Input: Work.Billed

	Name	DueDate	Amt
1	Tom	10OCT2009	1000
2	Dick	01JAN2010	123.45
3	Harry	24FEB2010	777.77

34

1: The Goal: Code to Generate

```
proc print data=late;  
run;
```

35

1: Program to Generate Code

* If more than 20 bills are more than 30 days over due, report them;

```
data late;  
  set billed end=final;  
  retain NLate 0;  
  if DueDate < today()-30 then do;  
    NLate = NLate + 1;  
    output;  
  end;  
  if final and NLate > 20 then  
    call execute('proc print data=late; run;');  
  drop NLate;  
run;
```

36

```

data late;
  set billed end=final;
  retain NLate 0;
  if DueDate < today()-30 then do;
    NLate = NLate + 1;
    output;
  end;
  if final and NLate > 20 then
    call execute('proc print data=late; run;');
  drop NLate;
run;

```

NOTE: There were 150 observations read from the data set WORK.BILLED.

NOTE: The data set WORK.LATE has 100 observations and 3 variables.

NOTE: DATA statement used (Total process time):

```

  real time    0.01 seconds
  cpu time     0.01 seconds

```

NOTE: CALL EXECUTE generated line.

1 + proc print data=late; run;

2 NOTE: There were 100 observations read from the data set WORK.LATE.

NOTE: PROCEDURE PRINT used (Total process time):

```

  real time    0.65 seconds
  cpu time     0.01 seconds

```

37

1: Output

Name	DueDate	Amt
Tom	10OCT2009	1000.00
Dick	01JAN2010	123.45
.	.	.

38

Questions?

How else could you solve this problem?

39

2: Data-based do loop

- Goal: Produce bar charts of holiday reservations at a restaurant
- Method:
 - Create data set with dates of holidays
 - Produce a chart of reservations for each holiday

2: Demonstrate executing multiple copies of the same code with some variations

40

2: Preparation: Part 1

* Create data set with dates of holidays;

```
data holidays;
  date=holiday('newyear',2010); output;
  date=holiday('valentines',2010); output;
  date=holiday('easter',2010); output;
  date=holiday('mothers',2010); output;
  date=holiday('memorial',2010); output;
  ...
  format date date9.;
run;
```

41

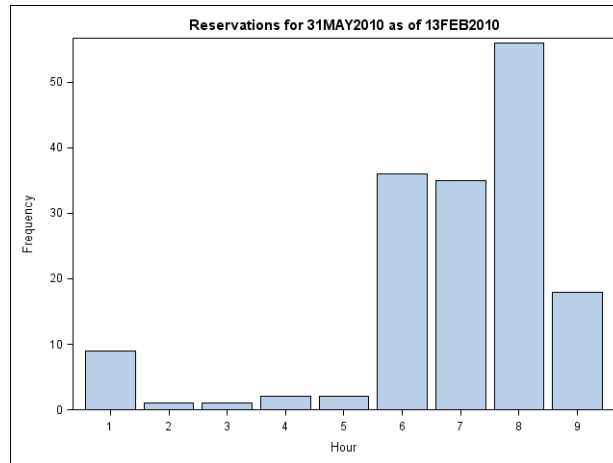
2: Preparation: Part 2

RSRV data set with reservations for all dates and times

Date	Hour	Booked
01JAN2010	1	7
01JAN2010	2	1
01JAN2010	3	1
01JAN2010	4	2
01JAN2010	5	8
	...	
02JAN2010	1	6
02JAN2010	2	1
02JAN2010	3	0
	...	

42

2: Sample Output



43

2: The Goal: Code to Generate

```

...
proc sgplot data=rsrv;
  where (date="31MAY2010"d);
  vbar hour / freq=booked missing ;
  title "Reservations for 31MAY2010 as of
    13FEB2010";
run;
...
proc sgplot data=rsrv;
  where (date="04JUL2010"d);
  vbar hour / freq=booked missing ;
  title "Reservations for 04JUL2010 as of
    13FEB2010";
run;
...

```

44

2: Concatenating 3 text strings to build:
where (date="01JAN2010"d);

Method 1:

```
' where (date=" ' ||
      put(date,date9.) ||
      '"d); '
```

Method 2:

```
catt(' where (date=" ' ,
      put(date,date9.) ,
      '"d); ' );
```

45

2: Program to generate one chart
For 01JAN2010 – without title

* Create a chart of reservations (by hour) for New Year's Day;

```
data _null_;
  date= "01jan2010"d;
  call execute('proc sgplot data=rsrv;');
  call execute('where (date=" ' ||
      put(date,date9.) ||
      '"d); ');
  call execute('vbar hour / freq=booked
      missing ; ');
  call execute('run;');
run;
```

Alternative: Use one very long call execute

46

2: Program to generate many charts

```
* Create a chart of reservations (by hour) for every holiday;
data _null_;
  set holidays;
  call execute('proc sgplot data=rsrv;');

  call execute('where (date=" ' ||
               put(date,date9.) || '"d); ');

  call execute('vbar hour / freq=booked
               missing ; ');

  call execute('run;');
run;
```

47

2: Program to generate many charts with titles

```
data _null_;
  set holidays;
  call execute('proc sgplot data=rsrv;');
  call execute ('where (date=" ' ||
               put(date,date9.) || '"d); ');
  call execute('vbar hour / freq=booked missing ; ');

  call execute('title "Reservations for ' ||
               put(date,date9.) ||
               ' as of ' ||
               put(today(),date9.) ||
               '"');

  call execute('run;');
run;
```

48

NOTE: CALL EXECUTE generated line.

```
1 + proc sgplot data=rsrv;
2 + where(date="01JAN2010"d);
3 + vbar hour / freq=booked missing ;
4 + title "Reservations for 01JAN2010 as of 13FEB2010";
5 + run;
6 NOTE: PROCEDURE SGPLOT used (Total process time):
   real time    0.14 seconds
   cpu time     0.06 seconds
```

NOTE: Listing image output written to SGPlot10.png.

NOTE: There were 9 observations read from the data set WORK.RSRV.
WHERE date='01JAN2010'D;

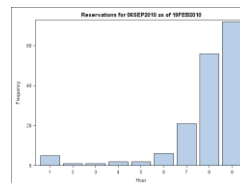
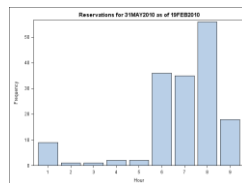
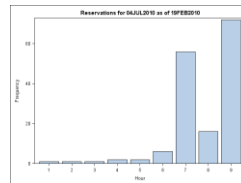
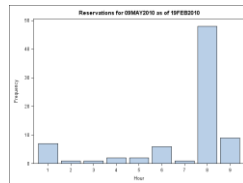
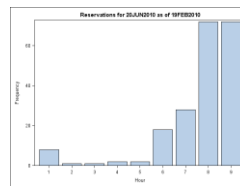
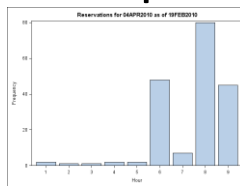
```
6 + proc sgplot data=rsrv;
7 + where(date="14FEB2010"d);
8 + vbar hour / freq=booked missing ;
9 + title "Reservations for 14FEB2010 as of 13FEB2010";
10 + run;
```

NOTE: PROCEDURE SGPLOT used (Total process time):

```
real time    0.10 seconds
cpu time     0.04 seconds
```

49

2: Sample Output



...

50

Questions?

How would you handle the “forest of quotation marks”?

51

Too many quotation marks can be hazardous to your sanity

“Very quickly one learns that it is a pain to spell out a lot of SAS code using CALL EXECUTE.”

Ian Whitlock



52

3: CALL EXECUTE with macros

```
call execute('%booked(date=' || date || ')');
```

3: Demonstrate executing a macro with different parameter values

53

3: Define macro

```
%macro booked(date= );

  proc sgplot data=rsrv;
    where (date="&date"d);
    vbar hour / freq=booked missing ;
    title "Reservations for &date as of
%sysfunc(putn(today(),date9.))";
  run;
  title;

%mend booked;
```

54

3: The Goal: Code to Generate

```
%booked(date= 01JAN2010);  
%booked(date= 14FEB2010);  
  
...  
  
%booked(date= 25NOV2010);  
%booked(date= 25DEC2010);
```

55

3: Program to generate many macro calls

```
data _null_;  
  set holidays;  
  call execute('%booked(date=' ||  
              put(date,date9.) || ')');  
run;
```

56

Questions?

What would happen if we used double quotes around a macro call in a CALL EXECUTE?

57

Rules for success with CALL EXECUTE

1. Use CALL EXECUTE with macros
This avoids difficulties with quoting.
2. Use single quotes around macro code.
Otherwise the code executes instantly, instead of end at the end of the step.
3. Use step boundaries like RUN.
Otherwise the timing of the execution of the generated code might not be as you intended.

58

4: Process data sets in a library

- Goal: Run same PROCs on all data sets with many observations in one data library
- Method:
 - Read metadata about all data sets in a library
 - Select data sets with many observations
 - Run PROC MEANS and PRINT for selected data sets

4: Demonstrate using a SAS DICTIONARY table as input to create macro calls with different parameter values

59

4: Metadata

- What are DICTIONARY tables?
 - Contain metadata about the active environment
 - Can only be used in PROC SQL
 - Tables can be processed like any other table
- How can you learn what metadata is available?

```
proc sql;  
  select * from dictionary.dictionaries;  
  select distinct memname from dictionary.dictionaries  
          order by memname;  
  describe table dictionary.tables;  
quit;
```

60

4: DICTIONARY.TABLES Contents

```
libname char(8) label='Library Name',  
memname char(32) label='Member Name',  
memtype char(8) label='Member Type',  
...  
crdate num format=DATETIME informat=DATETIME  
  label='Date Created',  
modate num format=DATETIME informat=DATETIME  
  label='Date Modified',  
  
nobs num label='Number of Physical Observations',  
obslen num label='Observation Length',  
nvar num label='Number of Variables',  
...
```

61

4: SASHELP.VTABLE

- What is SASHELP.VTABLE?
 - View of DICTIONARY.TABLES
 - Same columns / variables
 - Can be read in a DATA step, as if it were a data set

62

4: Define macro to create reports about each large data set

```
%macro QuickRpt(inputds=);  
  proc means data=&inputds;  
  run;  
  proc print data=&inputds(obs=10) label;  
  run;  
%mend QuickRpt;
```

63

4: The Goal: Code to Generate

```
%QuickRpt(inputds=SASHELP.BWEIGHT);  
%QuickRpt(inputds=SASHELP.FTABLE);  
%QuickRpt(inputds=SASHELP.PLFIPS);  
...
```

64

4: Program to generate many reports

```

data _null_;
  set sashelp.vtable;
  where libname="SASHELP" and nobs > 10000;
  length inputds $80;
  inputds=catt(libname, '.', memname);
  call execute('%QuickRpt(inputds= ' ||
               inputds ||
               '); ');
run;

```

65

4: Log

```

data _null_;
  set sashelp.vtable;
  where libname="SASHELP" and nobs > 10000;
  length inputds $80;
  inputds=catt(libname, '.', memname);
  call execute( '%quickrpt(inputds= ' || inputds || '); ');
run;
NOTE: There were 9 observations read from the data set
      SASHELP.VTABLE WHERE (libname='SASHELP') and (nobs>10000);
NOTE: DATA statement used (Total process time):
      real time           8.98 seconds
      cpu time            0.82 seconds
NOTE: CALL EXECUTE generated line.
1 + proc means data=SASHELP.BWEIGHT; run;
2 NOTE: There were 50000 observations read from the data set
      SASHELP.BWEIGHT.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.23 seconds
      cpu time            0.03 seconds
1 + proc print data=SASHELP.BWEIGHT(obs=10) label; run;
...

```

66

5: Checking Data

- Goal: Apply data-driven rules to check an input data set
- Method:
 - Create data set with rules
 - Generate PROC SQL with WHERE clauses to apply the rules

5: Demonstrate creating PROC SQL code

67

5: Sample Data

```
data LifeSpans;
  length Name $14 Birth Death 8;
  input  name birth death;
  format birth death date9.;
  informat birth death anydtdte.;
datalines;
Bach           7/28/1750      3/21/1685
Mozart         1/27/1756      12/5/1791
Beethoven     12/15/1770     3/26/1827
Brahms        5/7/1833       4/3/1897
Dvořák        9/8/1841       5/1/1904
Rachmaninoff  4/1/1873       3/28/1943
Prokofiev     4/27/1891      3/5/2953
;;;
run;
```

68

5: Rules Spreadsheet

TestDate	Relation	RefDate	Message
BIRTH	GT	DEATH	Birth after death
DEATH	GT	TODAY()	Death after today

69

5: Import Spreadsheet

```
proc import
    datafile= 'c:\checking\rules.xls'
    dbms=excel
    out=work.rules replace;
    getnames=yes;
run;
```

70

5: Work.Rules data set

VIEWTABLE: Work.Rules				
	TestDate	Relation	RefDate	Message
1	BIRTH	GT	DEATH	Birth after death
2	DEATH	GT	TODAY()	Death after today

71

5: The Goal: Code to Generate

```
proc sql;
select name, Birth, Death, "Birth after death "
from lifespans
where
BIRTH GT DEATH
order by name;
quit;
```

```
proc sql;
select name, Birth, Death, "Death after today "
from lifespans
where
DEATH GT TODAY()
order by name;
quit;
```

72

5: Program to generate many queries

```

data _null_;
  set work.rules end=eof;
  call execute( 'proc sql; select name,
               Birth, Death, "' || Message || "' );
  call execute( "  from lifespans");
  call execute( "  where ");
  call execute(testdate||relation||refdate);
  call execute( "  order by name; quit;");
run;

```

73

NOTE: CALL EXECUTE generated line.

```

1 + proc sql;
1 +   select name, Birth, Death, " Birth after death"
2 +   from lifespans
3 +   where
4 +   BIRTH GT DEATH
5 +   order by name;
5 +   quit;

```

NOTE: PROCEDURE SQL used (Total process time):

```

real time    0.25 seconds
cpu time     0.04 seconds

```

```

6 + proc sql;
6 +   select name, Birth, Death, " Death after today"
7 +   from lifespans
8 +   where
9 +   DEATH GT TODAY()
10 +  order by name;
10 +  quit;

```

NOTE: PROCEDURE SQL used (Total process time):

```

real time    0.01 seconds
cpu time     0.01 seconds

```

74

5: Output

Name	Birth	Death	
Bach	28JUL1750	21MAR1685	Birth after death

Name	Birth	Death	
Prokofiev	27APR1891	05MAR2953	Death after today

75

6: Run all programs in a directory

- Goal: Run all programs in a directory
- Method:
 - Read directory to get program names
 - Use CALL EXECUTE to issue %INCLUDE for each program

Yes, this example really uses BOTH techniques...

76

6: Get list of programs in a directory

- Use DIR command (or ls command in Unix)
- Use a PIPE receive the output from the DIR command
- Assign a file reference to the PIPE
- Use an INFILE statement in a DATA step to read from the file reference for the PIPE

77

6: Run DIR command in DATA step

```
filename pgmfile PIPE 'dir "C:\saspgms\";  
data _null_;  
    length pgmname $200;  
    infile pgmfile truncover;  
    input sasfile $ 1-200;  
    * ...;  
run;  
filename pgmfile clear;
```

78

6: Output from DIR command

```
Volume in drive C is Preload  
Volume Serial Number is A6B7-B172
```

```
Directory of C:\saspgms
```

```
02/06/2010  03:57 PM    <DIR>          .  
02/06/2010  03:57 PM    <DIR>          ..  
02/06/2010  03:57 PM    14 hidad.sas  
02/06/2010  03:57 PM    14 himom.sas  
.  
.  
.  
2 File(s)                28 bytes  
2 Dir(s)  98,133,864,448 bytes free
```

79

6: The Goal: Code to Generate

```
%INCLUDE "C:\saspgms\hidad.sas";  
%INCLUDE "C:\saspgms\himom.sas";  
...
```

80

6: Program to generate many includes

```
* Based on an example by Daniel Boisvert;
filename pgmfile PIPE 'dir "C:\saspqms\"';
data _null_;
  length pgmname $200;
  infile pgmfile truncover;
  input sasfile $ 1-200;
  if scan(sasfile,-1, '.')='sas';
  pgmname=scan(sasfile,-1, ' ');
  call execute('%include
               "C:\saspqms\' || pgmname || \';');
run;
filename pgmfile clear;
```

81

6: Log

NOTE: CALL EXECUTE generated line.

```
1  +  %INCLUDE "C:\saspqms\hidad.sas";
hi dad
2  +  %INCLUDE "C:\saspqms\himom.sas";
hi mom
. . .
```

82

Summary

- Use a macro when:
 - parameters can easily contain control information
- Use CALL EXECUTE when either:
 - control information is in data sets, spreadsheets, etc.
 - lots of control information is required
 - there are difficult special characters in control info
- Use %INCLUDE when:
 - to use code that is stored in files
 - multiple steps are needed to generate the code
 - when a DATALINES or CARDS statement is needed
 - PROC SQL is needed to generate the code

83

Sources

- Vergile, Bob, “Magic with CALL EXECUTE”, SUGI 22, 1997.
- Whitlock, H. Ian, “CALL EXECUTE: How and Why”, SUGI 22, 1997
- Michel, Denis “CALL EXECUTE: A Powerful Data Management Tool” SUGI 30
- Boisvert, Daniel and Shafi Chowdhury, Shafi, “CALL EXECUTE for everyone!” PharmaSUG 2006
- Ruelle, Alissa and Moses, Kitty, “CALL EXECUTE: A Primer”, PharmaSUG 2006

84

Contact Information

Paulette Staum
Paul Waldron Consulting
2 Tupper Lane
West Nyack, NY 10994
staump@optonline.net

85

Appendix CALL EXECUTE timing demo

- SAS non-macro statements execute **after** a step boundary.
- SAS macro statements, within single quotes, that resolve to SAS statements, execute **after** a step boundary.

86

CALL EXECUTE timing

- Pure macro statements
 - within single quotation marks, resolve during program execution, just **before** step boundary.
 - within double quotation marks, resolve while the DATA step is being constructed [which is pointless, so don't bother!].

87

Demo of Timing

```
data _null_;
  putlog "#1 from first putlog in step";

  call execute('data _null_; putlog "#2 from call
  execute putlog"; run;');

  call execute('%put #3 from call execute macro put in
  single quotes;');

  call execute("%put #4 from call execute macro put in
  double quotes;");

  putlog "#5 from last putlog in step";
run;
%put #6 from macro put after data step;
```

88

Demo Timing Log: Part 1

```

data _null_;
  putlog "#1 from first putlog in step";
  call execute('data _null_; putlog "#2 from call
execute putlog"; run;');
  call execute('%put #3 from call execute macro
put in single quotes;');
  call execute("%put #4 from call execute macro
put in double quotes;");
#4 from call execute macro put in double quotes
  putlog "#5 from last putlog in step";
run;
#1 from first putlog in step
#3 from call execute macro put in single quotes
NOTE: DATA statement used (Total process time):

```

89

Demo Timing Log: Part 2

```

#5 from last putlog in step
NOTE: CALL EXECUTE generated line.
1 + data _null_; putlog "#2 from call execute
  putlog"; run;
2 #2 from call execute putlog
NOTE: DATA statement used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds
      %put #6 from macro put after data step;
#6 from macro put after data step

```

90