ABSTRACT

Often a production program needs to be slightly modified to run in an optional way. That program is copied and the copy is modified per the new specifications. Now both programs reside in production and one or the other is run depending on which option the user needs. One obvious drawback to this approach is when the common portion of the programs needs to be fixed or modified, both files require the same modifications. However, there is a technique to run the one program with the modifications/options as an input parameter.

SYSPARM is a base SAS® system option, which allows the user to pass information to the program from the operating system. SYSPARM is available for all operating systems, the syntax is slightly different for each operating system. For this paper, I will be using the UNIX syntax. This paper will be directed to the beginner/moderate skill level, basic knowledge of the macro language will be required.

INTRODUCTION

Programmers can write flexible code that produce slightly different reports using the same program. This prevents the need to maintain multiple programs that probably differ by only a few lines of code.

SYSPARM is a system option that is used in the host environment to pass information to the SAS® program. This information is now available to the SAS® program to use in a data step or in macro language.

To access the information in a data step use the sysparm() function. The value returned from sysparm() is a character string. To access the information as a macro variable use the automatic macro variable &sysparm. As with all macro variables, &sysparm, is also a character string.

EXAMPLES

The following are 6 examples using the sysparm option. Each example is structured as follows: (1) the SAS® code, (2) the UNIX command to submit the program, and (3) the resultant output. If the &sysparm macro variable is used then the resultant output is preceded by the resolved SAS® code with the resolved &sysparm macro variable in italics. The example commands to run SAS® use the UNIX syntax. At the end of this paper are examples of other operating systems syntax.

The following dataset “HEIGHT” is used for all examples:

<table>
<thead>
<tr>
<th>NAME</th>
<th>HEIGHT</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>69</td>
<td>inches</td>
</tr>
<tr>
<td>Sally</td>
<td>64</td>
<td>inches</td>
</tr>
<tr>
<td>Peter</td>
<td>75</td>
<td>inches</td>
</tr>
</tbody>
</table>

EXAMPLE 1

Use the sysparm information as a conversion factor to print data in different units. This example uses the sysparm value 2.54 as the conversion factor from inches to centimeters.

```
data height;
  set height;
  if sysparm() ^= '' then
    height = height * input(sysparm(), best.);
run;
sas height.sas -sysparm 2.54
```

```
NAME   | HEIGHT      | UNIT 
------|-------------|------
John   | 175.26      | inches |
Sally  | 162.56      | inches |
Peter  | 190.50      | inches |
```

The advantage to this approach is that the program can run with any conversion factor, (e.g. use sysparm=.0833 to print in feet.) One problem, however, with this approach is that the output still shows “inches” as the unit of height.

EXAMPLE 2

Another approach is to pass a coded string to the program. Using the code you can also set the unit variable. In this example the value cm is being passed via the sysparm option. The program tests whether the value is cm then applies the proper conversion factor plus it assigns the unit variable.

```
data height;
  set height;
  if sysparm() = 'cm' then do;
    height = height * 2.54;
    unit='centimeters';
  end;
run;
sas height.sas -sysparm cm
```

```
NAME   | HEIGHT | UNIT  
------|--------|-------
John   | 175.26 | centimeters |
Sally  | 162.56 | centimeters |
Peter  | 190.50 | centimeters |
```

The disadvantage to this approach is that you would need to know all the possible conversion factors the user(s) would possibly use, and have separate sections to process each option (e.g. centimeter or feet conversions etc.)

EXAMPLE 3

This example combines the flexibility of Example 1 with the added information provided (i.e. assigning the unit variable) in Example 2. Assign the value 2.54|centimeters to the sysparm option. The pipe symbol serves to delimit two pieces of information, namely the conversion factor and the unit converted to. Now use the scan function to parse this string. The entire value needs to be enclosed in quotes to inform the operating system that everything within the quotes belongs to the sysparm option, otherwise UNIX will interpret this to “pipe” to the centimeters command which probably would result in an error. Of course you could use a different delimiter.
EXAMPLE 4
The sysparm information can also be accessed via the macro variable &SYSPARM. This next example shows how to subset or filter the data. In this case we are only looking for people who are taller than 65 inches.

```sas
data height;
set height;
%if &sysparm ^= %str( ) %then %do;
   if (&sysparm) then output;
%end;
run;
sas height.sas –sysparm 'height>65'
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>HEIGHT</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>69</td>
<td>inches</td>
</tr>
<tr>
<td>Peter</td>
<td>75</td>
<td>inches</td>
</tr>
</tbody>
</table>

Note the macro %if-%end construct surrounding the-data-step if statement, this is necessary in case the user didn’t use the sysparm option. So that you wouldn’t get an error when the &sysparm in the data-step-if statement resolved to blank. (i.e. if () then output.)

In this case the macro variable &sysparm resolves to height>65 which then is used in the data-step-if statement to subset the appropriate observations.

Also note that this section of code needs to be inside a macro since SAS® doesn’t allow %if-%end in open code. I have omitted the %macro and %mend statements in the above example for the sake of brevity in this paper.

Again note the quotes surrounding the height>65, this is so UNIX doesn’t interpret the greater than sign to redirect the output.

EXAMPLE 5
You can assign &sysparm to contain a complete SAS® statement, which would add even more flexibility to the program. This example requests the people that are taller than 65 inches not be printed.

```sas
data height;
set height;
&sysparm
run;
sas height.sas –sysparm 'if height>65 then delete;' 
data height;
set height;
   if height>65 then delete;
run;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>HEIGHT</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sally</td>
<td>64</td>
<td>inches</td>
</tr>
</tbody>
</table>

The macro variable &sysparm in this example resolves to the complete statement: if height>65 then delete; Notice that we don’t need the macro code %end construct here. This program would work fine if we were not using the sysparm option because the &sysparm macro variable would simply resolve to a blank statement and therefore all observations would be printed.

Also note that the sysparm parameter value contains the semi-colon which terminates the “if” statement and the usage of &sysparm does not contain a semi-colon in the program.

EXAMPLE 6
This last example shows how sysparm is used as a keyword to perform different sections of code (i.e. a proc freq or proc means.)

```sas
%if &sysparm = freq %then %do;
   proc freq data=height;
      tables height;
%end;
%else %if &sysparm = means %then %do;
   proc means data=height;
      var height;
%end;
run;
sas height.sas –sysparm 'means'
proc means data=height;
   var height;
run;
```

Analysis Variable : HEIGHT

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>Mean</td>
<td>Std Dev</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>69.3333</td>
<td>5.5075</td>
</tr>
<tr>
<td></td>
<td>Minimum</td>
<td>64.0000</td>
<td>75.0000</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>75.0000</td>
<td></td>
</tr>
</tbody>
</table>

This example requests that the means procedure be used to analyze this data.

Again note that this code needs to be inside a macro since the %if statements are not valid in open code. Again I have omitted the %macro and %mend statements for brevity.

OTHER CONSIDRATIONS
The maximum length of the string that the sysparm option can contain is 32,767. However if you need to use that much text in a sysparm parameter, you are probably better off writing a new program anyway.

SYNTAX FOR DIFFERENT OPERATING SYSTEMS
Syntax for the SYSPARM system option is host specific. The
following examples show the syntax for various operating systems. The examples assign the value XYZ to the &sysparm macro variable using the system option SYSPARM.

MVS // EXEC SAS, OPTIONS='SYSPARM=XYZ'
Batch // SYSIN DD DSN=program.sas, DISP=SHR
TSO sas input('''program.sas''')
opt('sysparm=XYZ')
CMS sas program.sas (sysparm=XYZ)
Open sas/sysparm=XYZ program.sas
VMS
UNIX, sas program.sas -sysparm XYZ
OS/2,
Windows

See the SAS® companion for your host system for more details.

CONCLUSION
Using the SYSPARM option is a way to write flexible SAS® code. Whether using the sysparm() function or the automatic macro variable &sysparm, one can write any number of constructs to obtain the desired results.

REFERENCES
SAS® Language Reference: Dictionary
SAS® Macro Language: Reference
SAS® Companion for the UNIX Environment
SAS® Companion for the Microsoft Windows Environment

ACKNOWLEDGMENTS
I would like to take this opportunity to thank my colleagues at Wyeth/Cambridge for their careful review and encouragement in preparing this paper and it’s accompanying presentation.

CONTACT INFORMATION (HEADER 1)
Contact the author at:
Steve Wong
Wyeth
35 Cambridge Park Drive
Cambridge, MA 02140
(617) 665-8874
Email: swong@genetics.com