ABSTRACT
Both the SAS/IntrNet® Application Dispatcher and SAS®9 Stored Processes translate name/value pair parameters in a Web-based request into SAS macro variables. This investigation presents four alternate strategies for handling the resulting SAS macro variables within the SAS code of these types of Web applications. In addition to a description of each strategy, a code segment will be shown along with a discussion of that strategy’s strengths and weaknesses. Of special interest will be how each strategy deals with a “non-response” user input or the situation where null values exist within multiple name/value pair input.

INTRODUCTION
Requests to invoke either SAS/IntrNet Dispatcher applications or SAS®9 Stored Processes typically have name/value pair parameters attached to them. These name/value pairs are translated into SAS macro variables that often need to be handled by the SAS code associated with these applications. In the case where the name/value pair involves a unique name, a single new SAS macro variable is created. The name of the SAS macro variable matches the name in the name/value pair and the value becomes the text assigned to the macro variable. In the case where a number of name/values share the same name, a set of new macro variables is created. This set includes a single macro variable using the first name/value pair; a macro variable using the name with a 0 appended to it, containing the number of name/values that share the name; and a list of names with numbers (starting with 1) appended to them, one for each of the matching name/value pairs. Consider an example of an HTML form, used to invoke a Web application, containing two checkboxes that share the same name:

```html
<input type="checkbox" name="tek" value="html" />
<input type="checkbox" name="tek" value="java" />
```

As noted, the SAS macro variables that get generated depend on the web request. In the checkbox example, this is based upon the user’s interaction with the HTML form elements. A single selection would generate 1 SAS macro variable. Selecting both checkboxes would generate 4 SAS macro variables. However, it might also be the case that the user would not choose either selection and not generate anything. For the checkbox example, consider the possible user actions and resulting macro variables:

<table>
<thead>
<tr>
<th>name/value Pairs in Web Request</th>
<th>Resulting SAS Macro Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>no input 1</td>
<td></td>
</tr>
<tr>
<td>one selection made 2</td>
<td>tek=html</td>
</tr>
<tr>
<td>both checkboxes selected</td>
<td>tek=html&amp;tek=java</td>
</tr>
</tbody>
</table>

Table 1. Generated SAS Macro Variables

1 nothing passed and no SAS macro variables created
2 values assume that the first checkbox is selected

The following sections investigate 4 different strategies for handling the possible SAS macro variable input. All of the section example code segments assume the use of the checkbox example presented earlier. As such, the discussion will involve the SAS macro variables tek, tek0, tek1, and tek2.

It is also important to note that all of the strategies presented will, using macro level programming, successfully capture “typical” single and multiple situations. Nevertheless, there are two special case scenarios that can cause problems and need to be considered. The first case involves differentiating a “non-response” or no input vs. a single null value input. The second case involves processing a set of like-named SAS macro variables where the first value is null.
STRATEGY 1 - THE %GLOBAL METHOD

The first strategy is the most commonly used and documented method [1][2]. In this strategy, using the checkbox example, the values for tek and tek0 are tested. A null value for tek would indicate that there is no input. If tek is not null then a subsequent test that tek0 is null would be done to check for single value input. If both tek and tek0 are not null then multiple input is assumed. The code for strategy 1 is shown below:

```sas
%MACRO check ;
    %GLOBAL tek tek0 ;
    %IF &tek EQ %THEN %DO ;
    %PUT NOTE: .. NO VALUE FOR tek ;
    %END ;
    %ELSE %IF &tek0 EQ %THEN %DO ;
    %PUT NOTE: .. ONE NAME/VALUE PASSED ;
    %PUT NOTE: .. tek = &tek ;
    %END ;
    %ELSE %DO ;
    %PUT NOTE: .. MULTIPLE NAME/VALUES PASSED ;
    %DO i = 1 %TO &tek0 ;
    %PUT NOTE: .. tek&i = &&tek&i ;
    %END;
    %END ;
    %PUT ;
%MEND ;
```

Note the use of a %GLOBAL statement to insure that the tek and tek0 macro variables exist. This provides an important safeguard for the conditional processing in the macro, but it is also the source of the first problem associated with this method. Consider the values for tek and tek0 show in table 2. Note that there is no difference between the values associated with no input vs. the values generated for a single null value input. If this distinction is important, an alternate strategy would be required.

Table 2. Strategy 1 - No Input vs. a Single Null Value

<table>
<thead>
<tr>
<th>name/value in Web Request</th>
<th>Resulting Value of tek0</th>
<th>Resulting Value of tek</th>
</tr>
</thead>
<tbody>
<tr>
<td>no input</td>
<td>tek0:</td>
<td>tek:</td>
</tr>
<tr>
<td>one null value</td>
<td>tek=</td>
<td>tek0:</td>
</tr>
</tbody>
</table>

1 nothing passed, macro variables created by %GLOBAL

The other problem that can arise with the use of this strategy is again associated with null values. Consider the values of the macro variables shown in table 3. The second set of macro variable values could occur if both checkboxes are selected and the value attribute for the first checkbox is coded as a null (value=""). Unfortunately, this pattern of values would be caught by the first macro level check (%IF &tek EQ %THEN) and generate the same results as no input or a single null value input situation – not a preferred outcome.

Table 3. Multiples With a First Value of Null

<table>
<thead>
<tr>
<th>name/value Pairs in Web Request</th>
<th>Resulting SAS Macro Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>both checkboxes selected</td>
<td>tek=html&amp;tek=java</td>
</tr>
<tr>
<td></td>
<td>tek:html</td>
</tr>
<tr>
<td></td>
<td>tek0:2</td>
</tr>
<tr>
<td></td>
<td>tek1:html</td>
</tr>
<tr>
<td></td>
<td>tek2:java</td>
</tr>
<tr>
<td>both checkboxes selected, but first value is null</td>
<td>tek=&amp;tek=java</td>
</tr>
<tr>
<td></td>
<td>tek0:2</td>
</tr>
<tr>
<td></td>
<td>tek1:</td>
</tr>
<tr>
<td></td>
<td>tek2:java</td>
</tr>
</tbody>
</table>
Consequently, the advantages of this method are that it is commonly used, well documented, and easy to code. Unfortunately, this method does not catch the distinction between no input and a single null value input; and it does not catch and process a set of macro variables if the first value passed is null.

STRATEGY 2 - PASSING THE COUNT
The second strategy involves inserting an initial counter into the web application request for each possible name that could generate multiple name/value pairs. In the checkbox example, a preset teK0=0 name/value would be added with the use of an HTML hidden field. Then, if there are no name/value pairs sharing the name teK, the initial counter will propagate a SAS macro variable with the default value of zero. Otherwise, if multiple name/value pairs are sent, a second version of the counter will get created that will replace the default value of the SAS macro variable with a value of two or more. One important restriction to note is that the hidden field containing the initial counter must be located physically in the form before the checkbox input tags:

```html
<input type="hidden" name="teK0" value="0" />
<input type="checkbox" name="teK" value="html" />
<input type="checkbox" name="teK" value="java" />
```

If this restriction is ignored, the final version of the SAS macro variable teK0 will always contain the default value of zero. The code for strategy 2 is shown below.

```sas
%Macro check ;
 %GLOBAL teK ;
 %if &teK0 eq 0 %then do ;
   %if &teK eq %then %put note: .. no value for teK ;
   %else %do ;
    %put note: .. one name/value passed ;
   %end ;
 %end ;
%else %do ;
 %put note: .. multiple name/value passed ;
 %do i = 1 %to &teK0 ;
    %put note: teK&i = &&teK&i ;
 %end ;
 %put ;
%end ;
%Mend ;
```

The advantage for this method is that, unlike strategy 1, it effectively traps for single vs. multiple name/values. Unfortunately, like strategy 1, it does not differentiate between no input and a single null value. This strategy also requires additional name/value pairs containing default values to be present in a request. Furthermore, there are restrictions on the placement of the default name/values within the request.

STRATEGY 3 – USE DICTIONARY.MACROS
Within a SAS session, directly accessible in PROC SQL, there is a SAS data set named dictionary.macros that contains the names and associated values of all of the macro variables that are available. Strategy 3 uses this SAS data set to check on any macro variables of interest. The code for this strategy is divided into two SAS macro programs. The first macro (%lookup) applies a where clause to the dictionary.macros data set and counts the resulting number of observations. This count is then loaded into a new global macro variable named _count. The second macro program (%check) calls the first macro program and then uses _count. For the checkbox example, table 4 shows the possible values that could get loaded into _count.

<table>
<thead>
<tr>
<th>user action</th>
<th>where clause found</th>
<th>_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>no input</td>
<td>neither teK nor teK0</td>
<td>0</td>
</tr>
<tr>
<td>one selection made</td>
<td>only teK</td>
<td>1</td>
</tr>
<tr>
<td>both checkboxes selected</td>
<td>both teK and teK0</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4. Resulting Values For _count
Using The Macro %lookup
It is important with strategy 3 that macro variables created by a processing request not be added into the global symbol table with either a `%GLOBAL` statement (like strategy 1) or passed along from the request (like strategy 2). The code for strategy 3 is shown below.

```sas
%MACRO lookup(name) ;
  %LET name = %UPCASE(&name) ;
  %GLOBAL _count ;
  PROC SQL NOPRINT ;
  SELECT COUNT(*) INTO : _count
  FROM dictionary.macros
  WHERE scope EQ "GLOBAL" AND
    (name EQ "&name" OR name EQ "&name.0") ;
  QUIT ;
  %LET _count = &_count ;
%MEND ;

%MACRO check ;
  %lookup(tek) ;
  %IF &_count EQ 0 %THEN %DO ;
    %PUT NOTE: .. NO tek PARAMETER PASSED ;
  %END ;
  %ELSE %IF &_count EQ 1 %THEN %DO ;
    %PUT NOTE: .. ONE NAME/VALUE PASSED ;
    %PUT NOTE: .. tek = &tek ;
  %END ;
  %ELSE %IF &_count EQ 2 %THEN %DO ;
    %PUT NOTE: .. MULTIPLE NAME/VALUEs PASSED ;
    %DO i = 1 %TO &tek0 ;
      %PUT NOTE: .. tek&i = &&tek&i ;
    %END ;
  %END ;
  %PUT ;
%MEND ;
```

The advantages for this method are that it can differentiate between nothing sent and a null value; and it effectively traps for single vs. multiple name/values. The only disadvantage for this method is that it requires slightly more complex programming than either strategy 1 or strategy 2. The requirement to use SQL cannot really be considered a disadvantage. While the direct use of `dictionary.macros` does require SQL, there is a data view based on this SAS data set named `sashelp.vmacro` that can be used anywhere. The appendix contains a rewritten data step version of the macro `%lookup` that uses this view.

**STRATEGY 4 - THE %SYMGLOBL METHOD**

In SAS®, there is a new macro function called `%SYMGLOBL` that is designed to check for the existence of macro variables within the global symbol table. Strategy 4 is nothing more than a revision of strategy 3 where the conditional checking is done directly with `%SYMGLOBL` and the macro program `%lookup` is discarded. The code for this strategy is shown below:

```sas
%MACRO check ;
  %PUT ;
  %IF %SYMGLOBL(tek) EQ 0 %THEN %DO ;
    %PUT NOTE: .. NO tek PARAMETER PASSED ;
  %END ;
  %ELSE %IF %SYMGLOBL(tek0) EQ 0 %THEN %DO ;
    %PUT NOTE: .. ONE NAME/VALUE PASSED ;
    %PUT NOTE: .. tek = &tek ;
  %END ;
  %ELSE %DO ;
    %PUT NOTE: .. MULTIPLE NAME/VALUEs PASSED ;
    %DO i = 1 %TO &tek0 ;
      %PUT NOTE: .. tek&i = &&tek&i ;
    %END ;
  %END ;
  %END ;
```
There are several advantages to this strategy. This strategy is easy to understand and involves very simple code. This strategy can differentiate between no input and a null single value. This strategy also effectively traps for single vs. multiple name/value pairs. The only disadvantage is that this strategy cannot be implemented using earlier versions of SAS. The macro function %SYMGLOBAL requires SAS®9.

CONCLUSION
The best strategy to employ is clearly strategy 4. The %SYMGLOBAL function does require SAS®9, but the code is simple and it handles any input. If SAS®9 is unavailable, strategy 3, using the dictionary.macros data set, is the next best suggestion. Unless other client-side (ex: javascript) or server-side (ex: php, jsp) processing is utilized to help correctly handle all input scenarios, both strategy 1 and strategy 2 should be avoided.

REFERENCES

APPENDIX
The following is an alternative version of the macro program %lookup used in strategy 3. The code uses data step processing and the sashelp.vmacro data view instead of an SQL query and the dictionary.macros data set.

%MACRO lookup(name) ;
  %LET name = %UPCASE(&name) ;
  %GLOBAL _count ;
  %LET _count = 0 ;
  DATA _NULL_ ;
  SET sashelp.vmacro ;
  WHERE scope EQ "GLOBAL" AND
    (name EQ "&name" OR name EQ "&name.0") ;
  CALL SYMPUT("_count",trim(left(_N_))) ;
  RUN ;
%MEND ;

CONTACT INFORMATION
Please forward comments and questions to:

Don Boudreaux, Ph.D.
E-mail: don.boudreaux@sas.com