Paper 46-30

%htmlForm: An HTML Form and SAS/IntrNet® Code Generator

Don Boudreaux, Ph.D., SAS Institute Inc., Austin, TX
Keith Cranford, Office of the Attorney General, Austin, TX

ABSTRACT

Developing SAS/IntrNet Dispatcher applications frequently involves writing HTML forms and coding macro programs to use the information that is output. This paper presents a macro program called **%htmlForm** that helps automate that process. It uses a SAS data set as input to generate a document that contains an HTML form and an associated code file that contains a collection of SAS macros designed to check any of the possible name/value pairs that are sent to output by that form. Together, these files are considered a good "starting point" for developing HTML interfaces for SAS/IntrNet applications.

INTRODUCTION

Writing HTML interfaces for Web applications typically involves working with HTML forms. Forms can be created with products such as FrontPage and Dreamweaver or by directly coding HTML form tags. In either case, when there are a small number of simple forms involved, the amount of effort that's needed is minimal. However, when there is an increase in the number of forms, the complexity of the forms involved, or both, then the amount of effort that's needed can be considerable. In addition, when the forms are used to pass information into SAS/IntrNet Dispatcher applications other issues arise. The name/value pairs that are sent out of the forms are passed into these SAS programs as macro variables and associated text. Within the SAS/IntrNet Dispatcher applications, it is important to check that these macro variables exist and verify whether no value, a single value, or multiple values are received. Although this type of checking can be accomplished with macro programming, it can get complex. The **%htmlForm** macro is designed to use a SAS data set that contains form definition information and generate an initial HTML document and an associated file that contain SAS macro code, which is designed to use the information passed from that document. This will reduce the amount of time and effort needed to develop SAS/IntrNet Dispatcher applications.

FORM DEFINITION INPUT

Like the **CNTLIN** data set that is used with PROC FORMAT, the **%htmlForm** macro uses an input SAS data set that contains design information. The observations in this SAS data set define an HTML form by specifying, for each form element, the type of form element, the name of the form element, the value associated with it, and any text used to label the value. The type of form element is in an 8-byte character variable that is named **element_type**. This variable should contain only one of the following words as its value: **text**, **select**, **radio**, **checkbox**, **textarea**, or **hidden**. These words correspond to the HTML form elements that will be generated by the observation. The name of the form element is in another 8-byte character variable that is named **element_name.** This variable should contain only a single word made up of the letters a-z or an underscore. The value of the form element is in an 8-byte character variable named **element_value** that follows the same conventions as **element_name**. The text that's used to label a value is in a character variable of size 64 that is named **text**. The following code shows an example form definition input SAS data set built with a DATA step:

```
data work.defn_1 ;
   label element_type = 'Type of Form Element'
         element_name = 'Name of Form Element'
         element_value = 'Value'
         text = 'Text Label for Value' ;
   infile datalines truncover ;
   input element_type $ element_name $ element_value $ text $64. ;
datalines ;
text      who
select    educ  BA    Bachelors Degree
select    educ  MA    Masters Degree
select    educ  PHD   Doctorate
radio     exp   LT_5  less than 5 years
radio     exp   GE_5  5 or more years
checkbox  web   html  SAS/IntrNet
checkbox  web   java  SAS AppDev Studio
textarea  note
hidden    note  2005
;
```

Notice that the data definition lines that are associated with the text field and textarea form elements do not contain

any **element_value** or **text** information. These variables are not used by the **%htmlForm** macro to construct these types of form elements. This is also the reason that the hidden field line does not contain any **text** information. Also consider that this definition data set could just as easily have been created from variables in any existing SAS data source or parsed from an existing HTML file.

HTML FORM DOCUMENT

The **%htmlForm** macro creates an initial working template of an HTML form. The template includes a full set of HTML document tags (**<html>**, **<head>**, **<body>,** …) and, using the input SAS data set specifications, a fully functional HTML form. Other than the requested form elements, the form is created with a number of attribute defaults and additional tags. The **<form>** tag includes an action attribute with a null value and a method attribute with a value of **"GET"**. Three **<input>** tags with a type attribute of **"hidden"** follow the **<form>** tag. The first two **<input>** tags are designed to provide the name **_service** with a default value of **"default"** and the name **_program** with a default value of null. Both of these names are required for SAS/IntrNet Broker requests. The third **<input>** tag provides a value of **"128"** for the name **_debug**. This name/value parameter directs SAS/IntrNet to display the SAS log along with the application output that's generated by the request. Although these tags would eventually need to be modified, they are ideal for local functional testing. The HTML form is also coded to provide a Reset button and a Submit button. Shown below is the HTML document code that would be generated by **%htmlForm** using the example definition SAS data set that was presented earlier. Following the code, Figure 1 shows what this HTML document would look like when rendered by Internet Explorer.

```
<!-- Defn Data: work.defn_1 -->
<!-- html File: c:\test\form_1.html -->
<html>
<head>
<title> %htmlForm </title>
</head>
<body>
<fieldset>
<legend><b> c:\test\form_1.html </b></legend>
<form action="" method="GET" >
<input type="hidden" name="_service" value="default" />
<input type="hidden" name="_program" value="" />
<input type="hidden" name="_debug" value="128" />
Label.text
<input type="text" name="who" value="" size="10" maxlength="10" />
<br/><br/>
Label.select <br/>
<select name="educ">
<option value="BA"> Bachelors Degree </option>
<option value="MA"> Masters Degree </option>
<option value="PHD"> Doctorate </option>
</select>
<br/><br/>
<input type="radio" name="exp" value="LT_5" /> less than 5 years <br/>
<input type="radio" name="exp" value="GE_5" /> 5 or more years <br/>
<br/>
<input type="checkbox" name="web" value="html" /> SAS/IntrNet <br/>
<input type="checkbox" name="web" value="java" /> SAS AppDev Studio <br/>
<br/>
Label.textarea <br/>
<textarea name="note" rows="2" cols="20"></textarea>
<br/><br/>
<input type="hidden" name="note" value="2005" />
<input type="reset" value="Reset " />
<input type="submit" value="Submit" />
</form>
</fieldset>
</body>
</html>
```

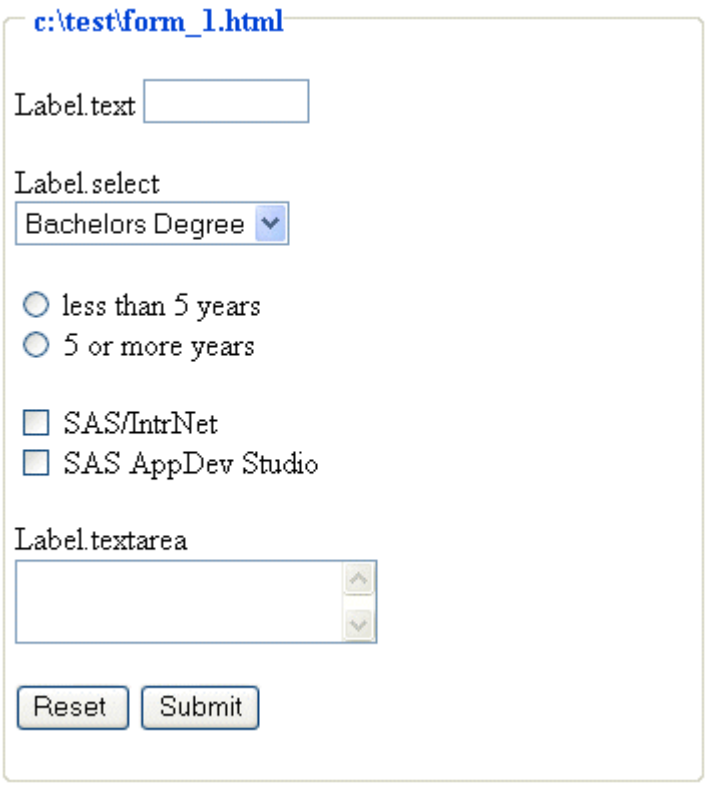

Figure 1. Example Document

Of course, to produce the final version of this form, the developer would want to replace any of the default element labels (in this example: **Label.text**, **Label.select**, and **Label.textarea**), provide any desired style features (such as backgrounds or font specifications), and include additional tags to control the layout of the form elements. In addition to modifying existing form elements, any new HTML content could also be included.

SAS/INTRNET APPLICATION BROKER CODE

The SAS/IntrNet Application Broker converts name/value pair output from an HTML source into SAS macro variables for use by SAS/IntrNet Dispatcher applications. In the cases involving a unique name, the name part of a name/value pair directly translates into a macro variable, and the value part is used as the text associated with that macro variable (even if the value is **null**). In the cases where multiple name/value pairs share a common name, a series of macro variables are generated and passed from the SAS/IntrNet Broker. This includes the initial name converted into a macro variable; a macro variable of that name with a 0 appended to it, which contains the number of name/value pairs that share that name; and a set of numbered macro variables, one macro variable for each of the matching named pairs. Consider the example form shown in Figure 1, if both checkboxes named **web** are selected by the user, then two name/value pairs would be output from the HTML form and four macro variables would be created by SAS/IntrNet Broker:

```
HTML Name/Value Pair Output:       web=html&web=java
Broker Generated Macro Variables:  web=html web0=2 web1=html web2=java
```

It is also possible that a form element might not output any name/value pair. Again using Figure 1, if neither checkbox is selected, then no name/value pair would be output. SAS/IntrNet Broker would not create any macro variable named **web,** and any SAS/IntrNet Dispatcher application that's coded to use a macro trigger by that name would generate an unresolved macro variable reference error. Therefore, for any potential parameter, it is necessary to determine, whether no value is passed, a single name/value is passed, or multiple name/values are sent. Fortunately, this can easily be accomplished with the use of macro-level conditional processing and the **%SYMGLOBL** macro function. This SAS®9 function checks for the existence of a macro variable within the global symbol table. Note that the check for multiple values is not always necessary. Based on the designed output capabilities of each type of form element and a check for name reuse between the elements specified in the form definition data set, **%htmlForm** automatically determines if this check is coded.

The following SAS/IntrNet Dispatcher Application code was generated by the **%htmlForm** macro for the example form definition. Each successive macro is associated with a form element name and checks whether no name/value, a single name/value, or (when needed) multiple name/values are passed. At the end of this code segment, a short DATA step is used to output a simple text message. This message keeps SAS/IntrNet from generating a warning message that no output is produced.

```
%* Defn Data: work.defn_1 ;
%* Code File: c:\bin\test\code_1.txt ;

%MACRO _who ;
   %PUT ;
   %IF %SYMGLOBL(who ) EQ 0 %THEN %DO ;
      %PUT NOTE: .. NO who PARAMETER PASSED ;
   %END ;
   %ELSE %DO ;
      %PUT NOTE: .. ONE NAME/VALUE PASSED ;
      %PUT NOTE: .. who = &who ;
   %END ;
   %PUT ;
%MEND ;
%_who ;
%MACRO _educ ;
   %PUT ;
   %IF %SYMGLOBL(educ ) EQ 0 %THEN %DO ;
      %PUT NOTE: .. NO educ PARAMETER PASSED ;
   %END ;
   %ELSE %IF %SYMGLOBL(educ0 ) EQ 0 %THEN %DO ;
      %PUT NOTE: .. ONE NAME/VALUE PASSED ;
      %PUT NOTE: .. educ = &educ ;
   %END ;
   %ELSE %DO ;
      %PUT NOTE: .. MULTIPLE NAME/VALUEs PASSED ;
      %DO i = 1 %TO &educ0 ;
         %PUT NOTE: educ&i = &&educ&i ;
      %END ;
   %END ;
   %PUT ;
%MEND ;
%_educ ;
%MACRO _exp ;
   %PUT ;
   %IF %SYMGLOBL(exp ) EQ 0 %THEN %DO ;
      %PUT NOTE: .. NO exp PARAMETER PASSED ;
   %END ;
   %ELSE %DO ;
      %PUT NOTE: .. ONE NAME/VALUE PASSED ;
      %PUT NOTE: .. exp = &exp ;
   %END ;
   %PUT ;
%MEND ;
%_exp ;
%MACRO _web ;
   %PUT ;
   %IF %SYMGLOBL(web ) EQ 0 %THEN %DO ;
      %PUT NOTE: .. NO web PARAMETER PASSED ;
   %END ;
   %ELSE %IF %SYMGLOBL(web0 ) EQ 0 %THEN %DO ;
      %PUT NOTE: .. ONE NAME/VALUE PASSED ;
      %PUT NOTE: .. web = &web ;
   %END ;
   %ELSE %DO ;
      %PUT NOTE: .. MULTIPLE NAME/VALUEs PASSED ;
      %DO i = 1 %TO &web0 ;
         %PUT NOTE: web&i = &&web&i ;
```

```
      %END ;
   %END ;
   %PUT ;
%MEND ;
%_web ;
%MACRO _note ;
   %PUT ;
   %IF %SYMGLOBL(note ) EQ 0 %THEN %DO ;
      %PUT NOTE: .. NO note PARAMETER PASSED ;
   %END ;
   %ELSE %IF %SYMGLOBL(note0 ) EQ 0 %THEN %DO ;
      %PUT NOTE: .. ONE NAME/VALUE PASSED ;
      %PUT NOTE: .. note = &note ;
   %END ;
   %ELSE %DO ;
      %PUT NOTE: .. MULTIPLE NAME/VALUEs PASSED ;
      %DO i = 1 %TO &note0 ;
         %PUT NOTE: note&i = &&note&i ;
      %END ;
   %END ;
   %PUT ;
%MEND ;
%_note ;
DATA _NULL_ ;
   FILE _webout ;
   PUT "<hr/>" ;
   PUT "An htmlForm( ) Macro Generated <br/>" ;
   PUT "SAS/IntrNet Dispatcher Application" ;
   PUT "<hr/>" ;
RUN ;
```

A number of these macro programs only check whether no parameter passed or a single name/value pair was passed. For the example form, the form elements that are involved include the text field named **who** and the radio button group named **exp**. The macro programs for the other form elements are all set for possible multiple name/value occurrences. By default, any selection list or any name that is shared between elements will need to be coded for possible multiple values. This would include the selection list named **educ**, the two checkboxes that share the name **web**, and the textarea and hidden field that share the name **note**.

%HTMLFORM MACRO DEFINITION

The following subsections show the programming code that defines the **%htmlForm** macro that was used to create the previously shown example HTML form and associated SAS/IntrNet Dispatcher Application code.

SECTION 1 - LOADING THE INPUT DATA SET

Initially, the program code names the macro and defines a set of parameters: **data, html**, and **code**. The **data** parameter contains the name of the form definition input SAS data set. This SAS data set is assumed to exist and be appropriately structured (see the “FORM DEFINITION INPUT” section). The **html** parameter contains the name of the html document that the macro is expected to create. The **code** parameter gives the name of the text file that will contain the macro code needed to check the form output. The code in the macro begins by reading the form definition SAS data set into the SAS data set **FORM_INPUT** and validates the characteristics of the variable values. Specifically, the code filters the variable **type** to be a single word in lowercase, filters the variable **name** to be a single word in lowercase that meets the naming convention for SAS variables, filters the variable **value** to be a single word (case unaltered), and builds a variable named **obs_number** to retain the original order of the input definition statements. Two steps are then invoked to obtain and retain the **first.name** and **last.name** information of the form element names. This information is used to get value counts within elements and for checking name reuse between form elements later. Then, a DATA step is used to create a variable named **id** that identifies the order of the element names in the original form definition SAS data set and a PROC SORT step is used to return **FORM_INPUT** to that order.

```
%macro htmlForm(data=work.defn, html=form.html, code=code.txt) ;
   data FORM_INPUT(keep=type name value text obs_number) ;
      set &data ;
      length type name value $ 8 ;
```

```
      type = lowcase(scan(element_type,1)) ;
      name = lowcase(scan(element_name,1)) ;
      name = compress(name,,"nk") ;
      if anydigit(name,1)=1 then substr(name,1,1) = "x" ;
      select(type) ;
         when ("text","textarea") do ;
            value = " - " ;
            text = " - " ;
         end ;
         when ("hidden") do ;
            value = scan(element_value,1) ;
            text = " - " ;
         end ;
         otherwise value = scan(element_value,1) ;
      end ;
      obs_number+1 ;
   run ;
   proc sort data=FORM_INPUT ;
      by type name ;
   run ;
   data FORM_INPUT ;
      set FORM_INPUT ;
      by type name ;
      first_name = first.name ;
      last_name = last.name ;
   run ;
   proc sort data=FORM_INPUT ;
      by obs_number ;
   run ;
   data FORM_INPUT ;
      set FORM_INPUT ;
      if first_name=1 then id+1 ;
   run ;
```

SECTION 2 - CREATING THE HTML DOCUMENT

The second section of code writes the document that contains the requested HTML form using a set of DATA steps. The first DATA step writes the initial HTML document tags through the beginning of the HTML form definition. The second DATA step uses **FORM_INPUT** to write the tags for the elements in the form. Notice the defaults that are set for each type of form tag. The third DATA step provides the tags for the Reset and Submit buttons, the end of the form, the end of the body section, and the end of the document. A final DATA step writes the contents of the created HTML file into the SAS log.

```
data _null_ ;
   file "&html" ;
   put "<!-- Defn Data: &data -->" ;
   put "<!-- html File: &html -->" ;
   put '<html>' ;
   put '<head>' ;
   put '<title> %htmlForm </title>' ;
   put '</head>' ;
   put '<body>' ;
   put '<fieldset>' ;
   put "<legend><b> &html </b></legend>" ;
   put '<form action="" method="GET" >' ;
   put '<input type="hidden" name="_service" value="default" />' ;
   put '<input type="hidden" name="_program" value="" />' ;
   put '<input type="hidden" name="_debug" value="128" />' ;
run ;
data _null_ ;
   file "&html" mod ;
   set FORM_INPUT ;
   length qtype qname qvalue $ 16 ;
   qtype ='type=' !! quote(trim(type)) ;
```

```
      qname ='name=' !! quote(trim(name)) ;
      qvalue='value='!! quote(trim(value)) ;
      if value=" " then qvalue=compress(qvalue) ;
      select(type) ;
         when("select") do ;
            if first_name=1 then do ;
               put 'Label.select <br/>' ;
               put '<select ' qname +(-1) '>' ;
            end ;
            put '<option ' qvalue +(-1) '> ' text '</option>' ;
            if last_name=1 then do ;
               put '</select>' ;
               put '<br/><br/>' ;
            end ;
         end ;
         when ("radio", "checkbox") do ;
            put '<input ' qtype qname qvalue '/> ' text '<br/>' ;
            if last_name=1 then put '<br/>' ;
         end ;
         when ("text") do ;
            put 'Label.text ' ;
            put '<input ' qtype qname 'value="" size="10" maxlength="10" />' ;
            put '<br/><br/>' ;
         end ;
         when ("textarea") do ;
            put 'Label.textarea <br/>' ;
            put '<textarea ' qname 'cols="20" rows="2"></textarea>' ;
            put '<br/><br/>' ;
         end ;
         when ("hidden") do ;
            put '<input ' qtype qname qvalue '/> ' ;
         end ;
         otherwise ;
      end ;
   run ;
   data _null_ ;
      file "&html" mod ;
      put '<input type="reset" value="Reset " />' ;
      put '<input type="submit" value="Submit" />' ;
      put '</form>' ;
      put '</fieldset>' ;
      put '</body>' ;
      put '</html>' ;
   run ;
   data _null_ ; * write html doc to log ;
      infile "&html" ;
      input ;
      put _infile_ ;
   run ;
```

SECTION 3 - CHECK FOR MULTIPLES

Writing the HTML document is one of the primary goals of the **%htmlForm** macro, another goal is to write the macro code that's needed to check the output from that form. To accomplish this, it is necessary to determine, for each element in the HTML form, what is going to be output to the SAS/IntrNet Broker and given to a Dispatcher application. This determination starts by counting the number of values that are associated with each name and the element type (holding onto the name's id for ordering purposes). These counts are saved in a variable named **count** in a SAS data set named **FORM_OUTPUT**. This SAS data set is then read again by using a DATA step to search for multiple values within a given element name. The DATA step code looks at the value of **count** and considers what type of form element is involved. By default, selection lists are assumed to generate multiple name/values and radio button groups are assumed to generate a single name/value. Otherwise, a value greater than 1 for **count** is an indication that multiple name/values can be generated by that form element. An indicator variable named **mult_value** holds the results of this check. This DATA step also looks for name reuse between elements. Two elements that by themselves

would only generate a single name/value pair could generate multiple name/value pairs if they share the same element name. Consider the example form where the textarea and the hidden field are both named **note**. Checking for name reuse is accomplished by testing that the names that are associated with the counts are unique. An indicator variable named **name_reuse** holds the results of this determination. Then both indicator variables are used to create a single new composite named **check**. A non-zero value in **check** indicates that either multiple values are generated for a given element name, or the element name is reused by several form elements, or both conditions exist. Then **FORM_OUTPUT** is used to make another SAS data set called **FORM_MACROS** that contains a unique set of element names, a final indicator of multiple name/value existence across names called **multiple**, and the variable called **id** that preserves the order of the elements in the original input SAS data set.

```
proc sql ;
create table FORM_OUTPUT as
select name, type, count(*) as count, min(id) as id
   from FORM_INPUT
   group by name, type
   order by name ;
quit ;
data FORM_OUTPUT ;            /* 1st: check within name looking at output info */
   set FORM_OUTPUT ;
   by name ;
   * multiple name/value determination ;
   select (type) ;
      when ("select") mult_value=1 ;
      when ("radio")  mult_value=0 ;
      otherwise do ;
         if count GT 1 then mult_value=1 ;
         else mult_value=0 ;
      end ;
   end ;
   * name reuse determination ;
   name_reuse= NOT (first.name=1 AND last.name=1) ;
   * indicator for either condition ;
   check=(mult_value OR name_reuse) ;
run ;
proc sort data=FORM_OUTPUT ;
   by id ;
run ;
proc sql ;                    /* 2nd: check between names, make macro defn data */
create table FORM_MACROS as
select name, min(id) as id,
      case
         when sum(check) = 0 then 0
         else 1
       end as multiple
   from FORM_OUTPUT
   group by name
   order by id ;
quit ;
```

SECTION 4 - WRITE THE MACRO CODE

The code in this section begins with a DATA step that uses the variable **multiple**, from the SAS data set **FORM_MACROS**, to create the SAS/IntrNet Dispatcher Application code that would look for no value, a single value, or multiple values for each of the names in the HTML form. After this, one DATA step is used to generate a simple output message and another DATA step is used to write the contents of the macro definition code file into the SAS log. At the end of this section, a **%mend** statement ends the definition of the **%htmlForm** macro program.

```
data _null_ ;
   file "&code" ;
   set FORM_MACROS ;
   if _n_=1 then do ;
      put "%* Defn Data: &data ;" ;
      put "%* Code File: &code ;" / ;
   end ;
```

```
      mp = '_'!!name ;
      mv_name = '&'!!name ;
      select ;
         when (multiple EQ 0) do ;
            put '%MACRO ' mp ';' ;
            put '   %PUT ;' ;
            put '   %IF %SYMGLOBL(' name') EQ 0 %THEN %DO ;' ;
            put '      %PUT NOTE: .. NO ' name 'PARAMETER PASSED ;' ;
            put '   %END ;' ;
            put '   %ELSE %DO ;' ;
            put '      %PUT NOTE: .. ONE NAME/VALUE PASSED ; ' ;
            put '      %PUT NOTE: .. ' name '= ' mv_name ';' ;
            put '   %END ;' ;
            put '   %PUT ;' ;
            put '%MEND ;' ;
            put '%' mp ';' / ;
         end;
         when (multiple EQ 1) do ;
            name0 = trim(name)!!'0' ;
            mv_name0 = '&'!!trim(name0) ;
            m1_name = trim(name)!!'&i' ;
            m3_name = '&&'!!trim(name)!!'&i' ;
            put '%MACRO ' mp ';' ;
            put '   %PUT ;' ;
            put '   %IF %SYMGLOBL(' name') EQ 0 %THEN %DO ;' ;
            put '      %PUT NOTE: .. NO ' name 'PARAMETER PASSED ;' ;
            put '   %END ;' ;
            put '   %ELSE %IF %SYMGLOBL(' name0') EQ 0 %THEN %DO ;' ;
            put '      %PUT NOTE: .. ONE NAME/VALUE PASSED ; ' ;
            put '      %PUT NOTE: .. ' name '= ' mv_name ';' ;
            put '   %END ;' ;
            put '   %ELSE %DO ;' ;
            put '      %PUT NOTE: .. MULTIPLE NAME/VALUEs PASSED ;' ;
            put '      %DO i = 1 %TO ' mv_name0 ';' ;
            put '         %PUT NOTE: ' m1_name '= ' m3_name ';' ;
            put '      %END ;' ;
            put '   %END ;' ;
            put '   %PUT ;' ;
            put '%MEND ;' ;
            put '%' mp ';' / ;
         end ;
         otherwise ;
      end ;
   run ;
   data _null_ ;
      file "&code" mod ;
      put 'DATA _NULL_ ; ' ;
      put '   FILE _webout ; ' ;
      put '   PUT "<hr/>" ; ' ;
      put '   PUT "An htmlForm( ) Macro Generated <br/>" ; ' ;
      put '   PUT "SAS/IntrNet Dispatcher Application" ; ' ;
      put '   PUT "<hr/>" ; ' ;
      put 'RUN ; ' ;
   run ;
   data _null_ ; * write sas code to log ;
      infile "&code" ;
      input ;
      put _infile_ ;
   run ;

   %mend htmlForm ;
```

It is important to note that the macro code generated by **%htmlForm** provides, within individual macro programs, macro-level conditional checking for each form output element name. However, it does NOT provide for any

significant action that would be taken upon finding no parameter passed, a single name/value pair, or multiple set of values. Look carefully at the macro statements to be written into the code file. In all cases, the only result of any macro level checking is to write a message into the SAS log using **%PUT** statements. These macros only provide a "starting point" template for a SAS/IntrNet Dispatcher Application. The application programmer is responsible for modifying the code to generate any further specific processing beyond this.

CONCLUSION

The **%htmlForm** macro can help automate the work that's involved with creating HTML front-end interfaces for SAS/IntrNet applications. This macro provides a data-driven tool that generates an HTML form and a file that contains the SAS macro code to check that form's output. The **%htmlForm** macro will give the SAS/IntrNet Dispatcher Application developer a good bit of help and flexibility. It can also be easily modified to produce site-specific form and macro code.

DEVELOPMENT NOTES

The HTML tags that are generated by the **%htmlForm** macro follow the XHTML syntax specifications and were tested using Internet Explorer 6.0.28 under Windows XP Professional. SAS®9 is required for running the **%htmlForm** macro. This macro was written using several functions that are not available in older versions of SAS. Also, note that the macro, as shown, is not intended to represent production-level code. The **%htmlForm** macro does not perform any parameter validation, it contains no macro variable value checking, and provides only minimal functionality. Production-level code would also be subject to more extensive code review and testing than was performed on this macro.

RESOURCES

SAS Institute Inc. 2001. *SAS Web Tools: Advanced Dynamic Solutions Using SAS/IntrNet Software*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2004. *SAS Macro Language*. Cary, NC: SAS Institute Inc.

Web Technologies Community: SAS/IntrNet Software, SAS Institute Inc. Available http://support.sas.com/rnd/web/intrnet/.

Murdock, Kelly. 2000. *Master VISUALLY HTML 4 and XHTML 1*, Forest City, CA: IDG Books Worldwide, Inc.

CONTACT INFORMATION

Please forward comments and questions to:

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

Keith Cranford
E-mail: keith.cranford@cs.oag.state.tx.us