A Complete Guide to HTML Form Output for SAS/IntrNet® Developers

Don Boudreaux, SAS® Institute Inc., Austin, TX

ABSTRACT

A number of existing conference papers, course notes sections, and references can be found that describe the use of HTML form elements within SAS/IntrNet Broker applications. These sources usually provide several examples to illustrate how form element information gets passed into SAS code. Unfortunately, what is typically missing is a complete discussion of all the form elements and detailed description of their output. Which form elements only generate one name/value pair? Which form element can generate multiple name/value pairs? And, which form element does not pass any information at all? This paper will attempt to "fill the gaps" and discuss every HTML form element. This paper will also describe how HTML form output is processed through the Broker and present an example macro for processing the results. It is assumed that the reader is already comfortable with HTML coding, SAS/IntrNet concepts, and macro programming.

INTRODUCTION

HTML forms provide a number of different elements that allow for user input: radio buttons, checkboxes, text fields, textareas and selection lists. HTML forms also provide hidden fields for default input, reset buttons, and submit buttons. So, upon a submit (except for the reset button), these form elements can generate name/value pair output. Typically, the output will consist of the name of the element, an equal sign, and the value associated with the element. But, there are variations of this depending upon the form element involved and the user interaction that is processed. Consider, for example, a form element named educ that could possibly have the values "BA", "MA", or "PhD". Upon a submit, this form element might not generate any output all, a null value (educ=), a single name/value pair (educ=MA), or multiple name/value pairs (educ=BA&educ=MA). In the following discussion, we will consider each form element and note what type of output can be expected. We will also note how default values can be set.

HTML TAGS AND THEIR OUTPUT

Before beginning a detailed look at each individual form element, two assumptions need to be made. First, given that none of the form elements will generate any output without a name attribute, this discussion will assume that the name attribute is required and properly coded unless otherwise stated. Second, the name of any element can possibly show up in the output multiple times if its name is reused or shared by other elements - even if the elements using the same name cannot by themselves generate multiple name/value pairs. So, it will be assumed that the value of the name attribute is unique for different elements within a given form unless otherwise stated.

Note, all HTML shown in this paper follow the XHTML syntax specifications. All tag and attribute names are typed in lower case. All attributes must have a value that is defined within quotes. And, all tags have either a beginning and end tag or are designated as empty by ending with "/>". Testing of these HTML code segments was done under Windows XP Professional using Microsoft's Internet Explorer (version 6.0.28).

RADIO BUTTONS

Within an HTML form, an input tag with the attribute of type="radio" generates a radio button. The name attribute is required. Without it, nothing will be output from this form element and the selection behavior of the form element is lost (the user cannot change the state of the button). Typically, multiple radio buttons will share the same name. These like named buttons, called a radio button group, will then only be allowed to have a single selection between them. Selecting one member of a radio button group will deselect any other previous selection. Usually a value attribute is desired. Without it, a value of "on" is used as the default. The additional attribute of checked="checked" allows a radio button to be preselected. An example of a radio button group using the name exp is shown below:

```
<input type="radio" name="exp" value="LT_5"/>
<input type="radio" name="exp" value="GE_5"/>
```

For output, radio buttons can generate nothing, a null value, or a single name/value pair. If the user hits the submit button without one of the radio buttons being selected, then nothing will be output. Otherwise, once a radio button group has a selection, either by user interaction or by the use of the checked attribute in the tag definition, a single name/value pair will always be generated when a submit occurs. A null value is only possible if the value attribute is coded with an empty quoted value (value=""). Given that selecting any member of a radio button group would deselect all other members, multiple name/value pairs cannot be generated by this form element.

CHECKBOXES

Checkbox syntax also uses the input tag, but with the attribute of type="checkbox". The name attribute is required. Without it, nothing will be output from this form element. Usually a value attribute is desired. Without it, like radio

buttons, a value of "on" is used as the default. The additional attribute of checked="checked" allows a checkbox to be preselected. An example of a checkbox is shown below:

```
<input type="checkbox" name="web1" value="html"/>
```

For output, checkboxes can generate nothing, a null value, or a single name/value pair. If the user hits the submit button without checking a box, then nothing will be passed. For any selected checkbox, either by user interaction or by the use of the checked attribute in the tag definition, a single name/value pair will be generated in the output. A null value is only possible if the value attribute is coded with an empty quoted value (value=""). Individual checkboxes do not generate multiple name/value pairs. Although, it is not uncommon for a collection of checkboxes to use the same name and, as noted earlier, possibly generate multiple name/value pairs.

HIDDEN FIELDS

Hidden fields are created within an HTML form by using the input tag with the attribute of type="hidden". The name attribute is required to generate any output and an associated value attribute is typically used. And although these tags do not create any visual element within the HTML document, they are commonly used. This is especially true with SAS/IntrNet Broker applications - where the parameters _service, _program, and _debug are usually coded within hidden fields. An example of a hidden field is shown below:

```
<input type="hidden" name="SCSUG" value="2004"/>
```

If the name attribute is present, hidden fields will always generate a name/value pair upon submission. It is possible, but highly unusual, to get a null value output from this element. A null value is generated if either the value attribute is coded with an empty quoted value (value="") or the value attribute is not used.

RESET BUTTON

A reset button is created using the input tag with the attribute of type="reset". The name attribute is not required. The reset button will work with or without one. The HTML form will display a button for this element with a default label of "Reset". A value attribute can be used to set an alternate label. The example below just adds some spacing to the “Reset” label - making the button slightly wider:

```
<input type="reset" value=" Reset "/>
```

Nothing is output from this form element (even if a name attribute is added).

SUBMIT BUTTON

A submit button is created using the input tag with the attribute of type="submit". The name attribute is not required. The submit button works with or without one. By default, the HTML form will display a button for this element with a default label of "Submit Query". A value attribute can be used to set an alternate label. My example changes the button label to "Submit":

```
<input type="submit" value="Submit"/>
```

Unlike the rest button, this form element can generate output. If the name attribute is used, a single name/value pair will always be generated. The value will be either the default text of "Submit+Query" or any alternate text defined by the value attribute. And again, a null value is possible if the value attribute is coded as empty quoted text (value=""). Multiple name/value pairs are not possible with this element.

TEXT FIELDS

Text fields are created using an input tag with the attribute of type="text". The name attribute is required. Without it, nothing will be output from this form element. A default value of null is set unless a value attribute is coded. Typically a size attribute (display size) and maxlength attribute (max number of text characters allowed to be entered) are also included. The example shown below only displays and only allows 10 characters:

```
<input type="text" name="who" size="10" maxlength="10"/>
```

With the required name attribute, text fields will always output a single name/value with a null value - unless either alternate text is defined within the value attribute or the user types text into the field. Text fields, by themselves do not generate multiple name/value pairs.

In HTML there are two other tags that work almost exactly like the text fields. They are the password field (type="password") and the file selection field (type="file"). The only difference with password fields is that they "mask out" any default or user entered text. The file selection field is different in that it displays both a text entry area and an associated browse button.

TEXTAREAS

Textareas are like text fields with the exceptions that they allow multi-line entry into an area that looks like and acts

like a simple text editor. Textareas are defined with a textarea tag. This is a container tag that requires a beginning tag, an ending tag, and a name attribute. The default value associated with this tag is null, unless some initial text is inserted between the beginning and ending tags (there is no value attribute). In addition to the required name attribute, the cols and rows attributes are commonly used to control the display size of the textarea. Unlike text fields, there is no attribute that limits the amount of text that can be entered. An example textarea shown below would display enough room to type 2 rows of 20 characters.

```
<textarea name="note" cols="20" rows="2"></textarea>
```

Like text fields, textareas always generate a name/value pair. If no initial text is coded between the (beginning and ending) tags and the user does not enter any text into the textarea field, then a null value will be output. Otherwise a non-null value will be sent. Note that besides a word or group of words, a textarea value can contain line breaks. These show up in the output as a "%0D%0A" character sequence.

SELECTION LISTS

Selection lists (also called menus or selection menus) are defined with a select tag. Like textareas, this is a container tag that requires a beginning tag, an ending tag, and a name attribute. This tag is designed to allow for single or multiple selections using the multiple attribute. A size attribute can also be defined to control the number of selections that are shown. The choices allowed within a selection list are defined using imbedded option tags. These are also container tags that require a beginning and ending tag. These tags also typically contain a value attribute and are coded with display text between their beginning and ending tags. The value attribute of the selected option tag determines the text value used in the output. As described before, a null value can be coded into the value attribute. If the selected option does not have a value attribute, then the display text is used as the value. A null value will also be generated if there is no value attribute and no display text for the selected option. Any option can be preselected with the use of a selected attribute.

A single choice selection list is a selection list that allows only one selection. This is the default for selection lists that do not use the multiple attribute in their definitions. An example is shown below:

```
<select name="educ">
<option value="BA"> Bachelors Degree </option>
<option value="MA"> Masters Degree </option>
<option value="PhD"> Doctorate </option>
</select>
```

This list will display in the browser as a pull down menu and it will always return a name/value pair. By default, if a choice is not preset or selected by user interaction, the value of the first option tag will be output. But, this behavior can be changed by adding a size attribute (with a value greater than "1") to the select tag. Consider this example:

```
<select name="educ" size="3">
<option value="BA"> Bachelors Degree </option>
<option value="MA"> Masters Degree </option>
<option value="PhD"> Doctorate </option>
</select>
```

This selection list will display as a list box containing 3 choices. If a choice is not preset or selected by user interaction, it will not output anything! The default of using the value of the first option tag is turned off. However, once a choice is selected, one selection will always show and a single name/value pair will always be output. A single choice selection list will not generate multiple name/value pairs.

A multiple choice selection list is created when the attribute multiple is added to the select tag. It always displays as a list box. The default number of choices shown is four. This can be changed with the size attribute. If the number of choices exceeds the default or the value set by the size attribute, then a scroll bar is added to the box. An example of a multiple selection list is shown below:

```
<select name="educ" multiple="multiple">
<option value="BA"> Bachelors Degree </option>
<option value="MA"> Masters Degree </option>
<option value="PhD"> Doctorate </option>
</select>
```

Like a single selection list with a size attribute, if a choice is not preset or selected by user interaction, it will not output anything! Unlike a single selection list, multiple choices can be made and (using the control key) all of the choices can be unselected. So, multiple selection lists can output nothing at all, a single name/value pair, or multiple name/value pairs. Any (or all) of the values output could be null.

HTML OUTPUT SUMMARY

Table 1 gives a quick summary for all of the form elements. Do note, as mentioned earlier, that none of the form elements will output anything without a name attribute. Any of the tags that have a value attribute can be coded to generate a null. And, any set of these tags (that create output) could generate multiple name/value pairs - if they share the same name value.

Table 1.
Form Element Output

	nothing	null	single	multiple
Radio Button	yes/no	yes	yes	no
Checkbox	yes	yes	yes	no
Hidden Field	no	default	yes	no
Reset	yes	no	no	no
Submit	yes	yes	yes	no
Text Field	no	default	yes	no
Textarea	no	default	yes	no
Selection List	no	yes	yes	no
size=	yes	yes	yes	no
multiple=	yes	yes	yes	yes

SAS/INTRNET BROKER PROCESSING

The SAS/IntrNet Broker is a CGI tool that can be viewed as a web-based controller for batch processing SAS programs. Utilizing a browser, a user makes a request to run a prestored SAS program and a configured SAS server does the work. Inherent in this scenario, is the passing of information from an HTML page through the SAS/IntrNet Broker and then to SAS. The Broker passes this information into SAS as a collection of macro variables. The prestored SAS program, called a dispatcher application, can be nothing more than a simple collection of data and proc steps with embedded macro variables, but it is typically a macro program. As reviewed, HTML form elements can generate nothing, a null, single name/value pair, or multiple name/value pairs. In the case where nothing is passed from an HTML form, nothing is passed through the Broker. Using the educ example described in the introduction (and shown in the section on selection lists), consider the macro variable that would be generated by the Broker if a null value is output from the HTML form:

HTML Output: educ=
Broker: educ=

In the case where a single name/value is sent, the Broker sends the name/value information as a single macro variable and its associated value. Again, using the educ example described in the introduction, consider the macro variable that would be generated by the Broker if a single name/value is output:

HTML Output: educ=BA
Broker: educ=BA

Multiple name/value pairs sharing the same name cannot directly be translated into the same single macro variable. So, with multiple responses, the Broker creates a series of macro variables - a single macro variable with the first associated value (just like the single name/value case), a zero indexed named macro variable whose value is the number of responses, and a series of named macro variables indexed from one to the number of name/value pairs sent. Note the four macro variables created by the Broker for the two name/value pairs output from HTML:

HTML Output: educ=BA&educ=MA
Broker: educ=BA educ0=2 educ1=BA educ2=MA

A possible problem can arise when using a set of multiple name/value pairs and their associated collection of macro variables. It is very important not to interpret the order of the values. If different form elements, using the same name, had created the multiple name/values, then the order would naturally have no meaning. But, even if a selection list with the attribute of multiple had created the multiple name/values, then order is still not meaningful with respect to the order in which the selections were made. The order of the values in the output, only reflect the order in which the option tags are listed within the select definition. The multiple name/value HTML output example shown above (and the four macro variables made by the Broker), could have been generated by a user selecting either the Bachelors Degree choice first and then Masters Degree or by selecting Masters Degree first and Bachelors Degree second. If the distinction of selection order is important, the form would either have to be changed to include additional HTML elements or augmented with additional programming capability (Javascript, Java, ASP, PHP, JSP, etc).

USING THE RESULTS IN SAS

On the SAS side, it is relatively easy with macro programming to use the macro variables sent by the Broker. The most commonly published algorithm starts by declaring all macro variable names that might get passed from the Broker in a %GLOBAL statement. And, if there is a possibility that a name might be associated with multiple name/value pairs, then also declare that name with a zero appended to it as well. This guarantees that any expected macro variables will exist and can be used without generating an "unresolved macro variable" error. Next, using macro level conditional statements, the algorithm checks the values of the macro variables. By checking the values for both a macro variable name and that name with an appended zero, it is possible to know to what kind of output was processed through the Broker. For example, again using educ, checking that the macro variable &educ is null, means either, nothing was output from the form (and the macro variable was created by the %GLOBAL) or a null value was sent. Note that the distinction, made when discussing HTML output, between no output and a null value is lost. If &educ is not null and &educ0 is null then only one name/value pair was processed. And, if &educ is not null and &educ0 is not null then multiple name/value pairs where processed. The code listed below is a simple example of a macro program designed to check educ for possible multiple values. Other than conditionally checking the possible macro variables created by the Broker, this code is designed only to print out messages to the log. However, it could easily be modified for more serious tasks.

```
%GLOBAL educ educ0 ;
%MACRO educMultCheck ;
    %IF &educ EQ %THEN %DO ;          /* educ EQ null */
        %PUT no value ;               /* sent: nothing or null */
    %END ;
    %ELSE %IF &educ0 EQ %THEN %DO ;   /* educ  NE null, educ0 EQ null */
        %PUT educ=&educ ;             /* sent: one value only */
    %END ;
    %ELSE %DO ;
        %DO i=1 %TO &educ0 ;          /* educ  NE null, educ0 NE null */
            %PUT educ&i = &&educ&i ;  /* sent: maultiple values */
        %END ;
    %END ;
%MEND ;
%educMultCheck
```

In the case where multiple name/values are not a consideration, a less complex macro could be used that only checks for null values.

```
%GLOBAL educ educ0 ;
%MACRO educOneCheck ;
    %IF &educ EQ %THEN %DO ;          /* educ EQ null */
        %PUT no value ;               /* sent: nothing or null */
    %END ;
    %ELSE %DO ;                       /* educ  NE null */
        %PUT educ=&educ ;             /* sent: one value only */
    %END ;
%MEND ;
%educOneCheck
```

In both cases, remember to insure that the macro variables involved exist (by using the %GLOBAL statement) and be sure to execute the macro with an associated macro trigger. In a real application, it would also be common to check the values generated by an HTML form. If this is not done on the client side (with Javascript, a Java applet,

etc…), it can be done on the server side with additional macro level coding.

If the distinction between no value sent and a null value is important, then design the application interface to avoid HTML elements that could generate both. Another alternative would be to use the SQL dictionary.macros table to check for the existence of macro variables that match the names of the form elements. This would allow checking the no value sent situation, before considering a null, single, or multiple value condition. But, the names of the expected form elements would have to be hard coded into the dispatcher application, read in from a data source, or passed (with hidden fields) from the form itself. This would also assume that any macro variable names that are being passed from the Broker are deleted from the global symbol table between dispatcher applications.

CONCLUSION

Effectively building a SAS/IntrNet Broker application involves a great deal of work. On the front end of these types of applications, past using simple hyperlinks, the user interfaces usually involve coding HTML forms. Having a detailed understanding of all the form elements and their output capabilities is crucial to this effort. On the back end, knowing how HTML output passes through the Broker and how to process it within SAS is essential for properly using that information.

REFERENCES

[1] SAS Institute Inc. (2001), SAS Web Tools:
Static and Dynamic Solutions Using SAS/IntrNet Software,
Course Notes - Book Code 57695
Cary, NC: SAS Institute Inc.

[2] SAS Institute Inc. (2001), SAS Web Tools:
Advanced Dynamic Solutions Using SAS/IntrNet Software,
Course Notes - Book Code 57696
Cary, NC: SAS Institute Inc.

[3] Pratter, Frederick (2003), Web Development
with SAS by Example
Cary, NC: SAS Institute Inc.

[4] Castro, Elizabeth (2003), HTML For The World Wide Web,
Fifth Edition, with XHTML and CSS: Visual QuickStart Guide
Berkeley, CA: Peachpit Press

[5] McGrath, Mike (2003), XHTML
Warwickshire, United Kingdom: Computer Step.

ACKNOWLEDGMENTS

Special thanks to my dear wife for supporting my continued desire to attend conferences and for proof reading my writing.

CONTACT INFORMATION

Please forward comments and questions to:

Don Boudreaux, Ph.D.
SAS Institute Inc.
11920 Wilson Parke Ave
Austin, TX. 78726
Phone: 512.258.5171 ext 3335
E-mail: don.boudreaux@sas.com

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. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.