Paper 134-27

TAGSET Your It! Using ODS MARKUP to Create Pre-Filled HTML Form Tags

Thomas E. Kunselman, California State University, Stanislaus

Abstract

ODS MARKUP is an addition to the Output Delivery System starting with SAS Version 8.2. MARKUP generates markup language output from tagsets shipped with SAS or user defined tagsets created with PROC Template. This paper describes a subset of the PROC TEMPLATE code used to generate HTML form tags containing pre-filled data output from PROC PRINT. Example SAS code with PROC PRINT and the ODS statement are also provided to demonstrate how to use ODS MARKUP. The resulting HTML form tags can be inserted into an existing ODS HTML output stream destined for a file or sent to _webout for use with the SAS/IntrNet application dispatcher. These two examples provide a brief introduction to customizing and using ODS MARKUP tagsets. In addition, this example provides another method for users of the SAS/IntrNet application dispatcher to create dynamically generated HTML form tags.

Background

When dynamically generating HTML for output from the SAS/IntrNet application dispatcher there usually comes a point where a decision about what exactly to generate needs to be made. HTML provides several ways to do this using form objects such as a select box, radio buttons, or check boxes. How can these form objects be dynamically filled to allow choices consistent with frefreshed data?

There are probably as many ways as there are SAS programmers to create and dynamically fill these HTML form tags. One proven method calls a generic SCL program passing parameters to dynamically create the desired form object containing the generated choices.[1] Unfortunately this adds another layer of complexity of programming and maintenance by requiring knowledge of SCL.

Another way to dynamically create HTML form tags might be to use the tried and true DATA _NULL_ to create the form object. Creating the DATA _NULL_ within a macro would allow the code to be reusable. But if we're going to do that, why not simplify the process even further and replace the DATA _NULL_ with a PROC PRINT and an ODS MARKUP TAGSET to create the form object.

Introduction

> **Markup Language**. *Noun*. A system (as HTML or SGML) for marking or tagging a document that indicates its logical structure (as paragraphs) and gives instructions for its layout on the page for electronic transmission and display
> - Merriam-Webster's Collegiate Dictionary

Available experimentally in SAS Software 8.2, ODS MARKUP is used to generate markup language in your SAS output. The definition for how to insert the markup language is created using PROC TEMPLATE and a DEFINE TAGSET. The tagset specifies how to imbed markup tags into SAS output. There are several SAS Institute supplied tagsets available with SAS 8.2 including CSV, DOCBOOK, LATEX, PYX, TROFF, WML, XML and several others. Existing tagsets can be modified or new tagsets can be created from scratch using PROC TEMPLATE. Several user-defined tagsets, such as SYLK and XHTML, can be found in the SAS Institute WWW document, *Creating Customized Tagsets to Use with ODS and XML LIBNAME Engine*.[2] This reference also provides all of the information required to create customized tagsets.

Listing Tagsets and Source

Many of the existing tagsets did not ship with SAS 8.2 but instead are available for download on the SAS Institute web-site.[2] To print a list of available tagsets on your system, run the following:

```
PROC TEMPLATE;
   LIST TAGSETS;
RUN;
```

To see the source of a tagset run the following specifying the name of the tagset you want to review:

```
PROC TEMPLATE;
   SOURCE TAGSETS.HTML_SELBOX;
RUN;
```

The source for the HTML_SELBOX tagset can be viewed in Example 1 with a brief description about how the tagset works for those interested in creating their own custom tagsets.

HTML_SELBOX Custom Tagset Definition

Tagset definitions use events to embed markup tags into SAS output. These can be standard ODS events triggered by procedures or they can be user defined events. In both cases the DEFINE EVENT statement specifies what actions are to take place when the event is triggered. An event can also trigger another event using the TRIGGER event_name statement. To obtain a list of ODS events and variables generated by a procedure the EVENT_MAP tagset can be used. The EVENT_MAP XML output is useful in creating custom tagsets.

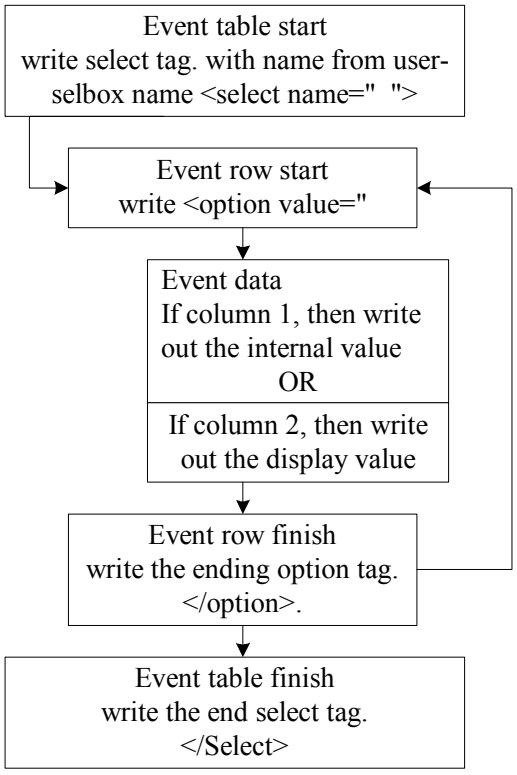
Example 1

```
proc template;
path sashelp.tmplmst(update);
define tagset tagsets.html_selbox;
notes "Tagset to generate an HTML Form
Select Box";
nobreakspace=' ';
output_type='HTML';
indent=2;
```

```
define event table;
mvar user_selboxname;
start:
   put "<select name="" user_selboxname
""">" CR;
    notes "Comment out the Column Head-
ers";
    put "<!";
finish:
   xdent;
   put "</select>" CR;
end;

define event row;
start:
   ndent;
   put "<option value=";
finish:
   put "</option>" CR;
   xdent;
end;

define event data;
   putq VALUE / if cmp( COLSTART , "1" );
   put ">" / if cmp( COLSTART , "1" );
   put VALUE / if cmp( COLSTART , "2" );
end;

end;
run;
```

HTML_SELBOX tagset flow

The events defined in the html_selbox tagset are table, table_head, row and data. Some of these events contain START and FINISH sections which indicate that the code is to be run either at the beginning or the end of the event. A stateless event, such as the DATA event in Example 1, has neither a start nor a finish section.



Creating a Custom Tagset

By using inheritance when creating a custom tagset, an existing tagset can be referenced as the parent in the DEFINE TAGSET statement using the PARENT= attribute. Any events defined in the new tagset supersede events with the same name in the parent tagset.

Because a tagset is used to create output, the majority of work in a tagset event is writing out tags or data. This is accomplished with PUT, PUTL and PUTQ. Each of these write data or text strings to the location specified in the ODS statement. PUTL writes a new line at the end and PUTQ will insert quotation marks around the value being output.

Conditional execution of an event statement can be achieved for any event statement. In Example 1 the event DATA uses conditional execution to determine whether to output the value with quotation marks or without. To use a conditional, insert a / at the end of the event statement followed by the conditional.

event statement / conditional;

Event conditions include: CMP to check for equality; EXISTS to ensure that a variable or a list of variables all contain values; ANY to check that at least one variable out of a list contains a value and; NOT to reverse a condition.

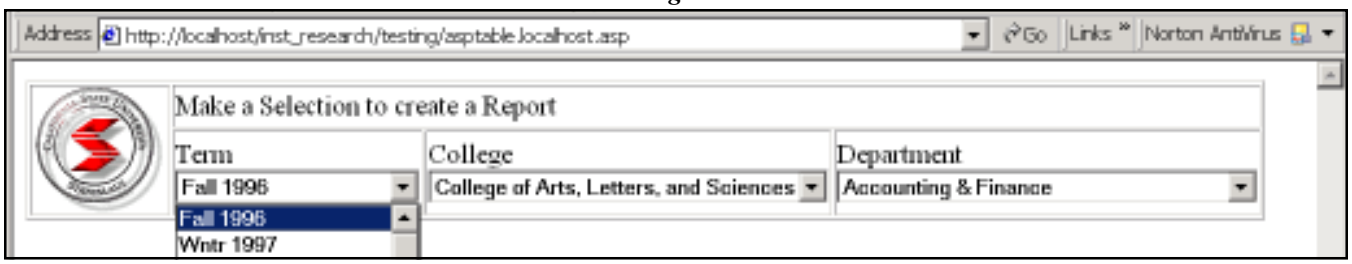
Example 1 also uses the MVARNAME statement to define a macro variable to be accessed within the tagset. In the html_selbox tagset the user_selboxname contains the name of the HTML select box to be used when creating the select box tags. The macro variable user_selboxname needs to be set before calling the tagset.

Using Custom Tagsets with ASP and SAS/IntrNet

Now that the HTML_SELBOX tagset has been defined the next step is to use it to replace the SCL program used to generate select boxes. The following example will use ASP to call the SAS/IntrNet application dispatcher and request that a program (sel_erss.sas) be run to generate the HTML select box. The application dispatcher will then return the HTML code generated by ODS using the html_selbox tagset. The resulting select boxes can then be used to make selections to subset data for a report to be dynamically generated.

The ASP code in Example 2 calls the application dispatcher passing the name of the program to run. Also sent is the data set name, webdata.summ_erss_unique. This data set is refreshed when the web data is refreshed and contains unique values for variables used to subset the larger data set. The other needed information items sent to the sel_erss.sas program are the name of the variable to create the select box options with, VARNAME, and the name of the select box which is sent in USER_SELBOXNAME. The application dispatcher will automatically convert all of these variable name and value pairs to macro variables for use in the SAS program.

Figure 1



The actual ASP file contains HTML and two additional sections of ASP code with calls to the sel_erss.sas program to generate a total of 3 select boxes using proc print and the html_selbox custom tagset. Figure 1 displays the browser results of the ASP document.

Example 2

```
<%
sub displayURLx(URL)
Dim AppServer, HTML
Set AppServer =
CreateObject("SAS.AppServerPostURL")

AppServer.webServer = "chomsky"
AppServer.URL = "/cgi-bin/broker.exe"
AppServer.queryString = URL

HTML = AppServer.openURL()

Response.Write HTML
End sub

displayURLx("_SERVICE=default
&_PROGRAM=webpgm.sel_erss.sas
&dataset=webdata.summ_erss_unique
&varname=term
&user_selboxname=selterm")
%>
```

The SAS program (sel_erss.sas) executed on the application dispatcher is listed in Example 3. The macro variables referenced contain the values passed from the ASP code in Example 2. The ODS MARKUP statement is used in the same way that ODS HTML would be used with the addition of the TYPE= parameter to identify the tagset to be used to create the output. Note that the macro variable USER_SELBOXNAME although not referenced in the program is used by the tagset with the macro variable name and value passed to the application dispatcher and assigned automatically in the sel_erss.sas program..

Example 3

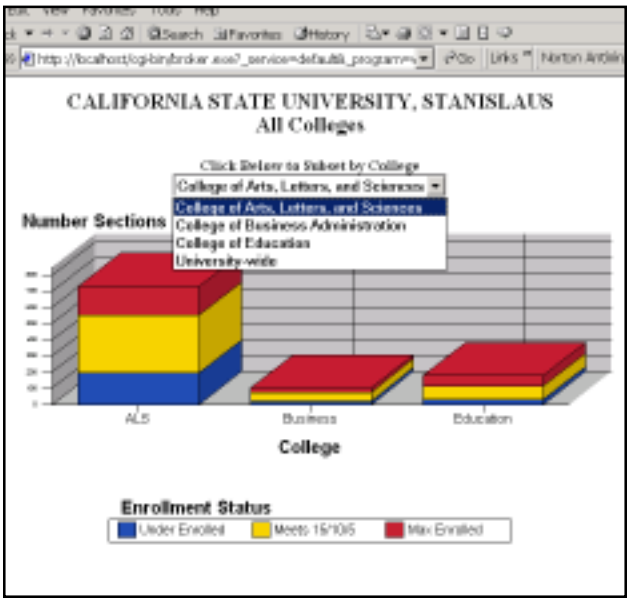
```
* sel_erss.sas;
ods markup type=html_selbox
            body=_webout;
 proc print data=webdata.&dataset noobs;
 where &varname._val ne '';
 var &varname._val &varname;
 run;
 ods markup close;
```

The HTML fragment in Example 4 is the select box generated from the sel_erss.sas program using the html_selbox tagset. The name of the select box has been inserted from the USER_SELBOXNAME macro variable. In addition, the first row of the PROC PRINT output contains the header information for the output table. The html_selbox tagset inserts a <! to comment out the first row of the table which contains the variable names who's values are used to generate the <option> tags. In this case TERM_VAL contains the unformatted value and TERM is the formatted value.

Example 4

```
<select name="selterm">
<!<option value="term_val">Term</option>
  <option value="199640">Fall 1996</option>
  <option value="199710">Wntr 1997</option>
  <option value="199720">Sprg 1997</option>
  <option value="199740">Fall 1997</option>
  <option value="199810">Wntr 1998</option>
  <option value="199820">Sprg 1998</option>
  <option value="199840">Fall 1998</option>
  <option value="199910">Wntr 1999</option>
  <option value="199920">Sprg 1999</option>
  <option value="199940">Fall 1999</option>
  <option value="200010">Wntr 2000</option>
  <option value="200020">Sprg 2000</option>
  <option value="200030">Summer 2000
(State)</option>
  <option value="20003X">20003X</option>
  <option value="200040">Fall 2000</option>
  <option value="20004X">20004X</option>
  <option value="200110">Wntr 2001</option>
  <option value="20011X">20011X</option>
  <option value="200120">Sprg 2001</option>
  <option value="20012X">20012X</option>
  <option value="200130">200130</option>
  <option value="200140">Fall 2001</option>
  <option value="200220">200220</option>
</select>
</td>
```

Figure 2



Dynamically filled form objects are very useful when providing access to data that is frequently updated. They provide a means for displaying all available choices without the necessity of manually updating static HTML pages on an hourly, daily, or weekly basis. Another application of dynamic form objects is in cases where access to data can be limited to subsets of the whole. In these cases a select box could be created that contains only values from the available subset.

Mixing Custom Tagsets with ODS HTML

While these tagsets can replace using a DATA _NULL_ to do the same thing, there may be times when there is a need to mix DATA _NULL_ or ODS HTML output with a custom tagset. In this example, a SAS/IntrNet application dispatcher program mixes HTML header and title information created with DATA _NULL_, a select box created with ODS MARKUP, and HTML generated from the DS2GRAF macro. The resulting output can be seen in Figure 2.

Example 5 contains the SAS program fragment used to generate the select box and the graph. In this particular example, the output is split into four different components specifying the titles and text, next the select box, then the graph macro and finally another DATA _NULL_ statement to write out the footer information. Because this is a dynamically generated application each of the ODS statements writes out to the _webout device.

The first ODS statement writes out the HTML header information with the option to tell ODS to write no closing HTML body tags. This allows the following ODS streams to concatenate their output to this first HTML stream.

The next ODS statement uses MARKUP and the HTML_SELBOX tagset to create a select box to provide a list of choices to select. Unlike the previous example where the name of the select box is passed to the application dispatcher, in this example the %let statement defines USER_SELBOXNAME for use by the html_selbox tagset. The TEST2 data set contains the CNTLOUT data set from PROC FORMAT for the college format used in this example. Not shown in this code fragment but needed to make the select box work is an HTML form tag. The HTML form should contain the select box and a submit button to send the selection to the application dispatcher.

The third component of this example is the %DS2GRAF macro which creates HTML tags to call the SAS Graph Applet. The PAGEPART=BODY option tells the macro to not create any HTML header or footer information but only what is needed to insert this output into an HTML page.

The final portion of the SAS code to write out footnotes and the HTML footer tags is not included here. The final section is similar to the first part of this example but instead uses the NO_TOP_MATTER parameter on the ODS HTML statement to output only the closing HTML document tags.[4]

Example 5

```
ods html body=_webout (dynamic
no_bottom_matter)
              path=&_tmpcat
(url=&_replay)
              rs=none
              style=styles.minimal
              ;
data _null_;
file _webout;
put "<CENTER>";
put "<H3>&csustan <BR> &drill_title";
put '<H5>Click Below to Subset by
College<BR>';
run;
ods html close;

%let user_selboxname = selcoll;
ods markup   type=html_selbox
             body=_webout;
proc print data=test2 noobs;
var valueText label;
run;
ods markup close;


%ds2graf(htmlfref=_webout,
              data=work.underenrolled,
              &set_chart_options
              pagepart=BODY,
              styleby=category,
```

The resulting dynamically generated HTML code fragment in Example 6 shows how ODS HTML and ODS MARKUP can be mixed to create a seamless customized output format including dynamically generated form objects. The header information from the first ODS HTML statement along with the titles created by the DATA _NULL_ have the <SELECT> tag appended. The select box name has been created from the macro variable by the tagset and following the <SELECT> tag is the <APPLET> tag containing the output from the %DS2GRAF macro.

Example 6

```
<HTML>
<!-- Generated by SAS Software -->
<!-- Http://www.sas.com -->
<HEAD>
<TITLE>SAS Output</TITLE>
<META http-equiv="Content-type" con-
tent=" charset=windows-1252">
</HEAD>
<BODY onload="startup()"
onunload="shutdown()">
<SCRIPT LANGUAGE="JavaScript">
<!--
// This script is to load all object
onLoad() functions
function startup(){ }
function shutdown(){ }
//-->
</SCRIPT>
<CENTER>
<H3>CALIFORNIA STATE UNIVERSITY,
STANISLAUS <BR> All Colleges
<H5>Click Below to Subset by College<BR>
```

```
<select name="selcoll">
<!<option value="valueText">LABEL</
option>
   <option value="'27' 'AS'">College of
Arts, Letters, and Sciences</option>
   <option value="'41' 'BU'">College of
Business Administration</option>
   <option value="'50' 'ED'">College of
Education</option>
   <option value="'99' 'UW'">University-
wide</option>
</select>

<APPLET CODE="GraphApplet.class"
     ARCHIVE="/sasweb/graph/graphapp.jar"
     ALT="SAS Institute Inc. GraphApplet"
```

Conclusion

ODS Markup is very powerful and flexible tool that can be used for dynamically generating output as well as for creating static output. Tagsets provide an alternative to using SCL or other methods for creating generic reusable methods for streaming data from the SAS/IntrNet application server and the learning curve is much less steep than that of SCL.

References

[1]http://www.psiconsultants.com/sas/tobe.htm (SUGI 2000) (Date: 10JAN2001 "To Be or Not to Be? That is the Dynamic Web Page Question.") Thomas Kunselman, Almira Lyst, and Donald Penix. SUGI 25.

[2]http://www.sas.com/rnd/base/topics/odstagsets (**Date: 17SEP2001 "Creating Customized Tagsets to Use with ODS and XML LIBNAME ENGINE**")

[3]http://www.sas.com/rnd/base/topics/templateFAQ/xmlf.htm (**Date: 17SEP2001 "XML, ODS and The Markup Language**")

[4]Sinking the Big Hole: Using SAS/IntrNet Application Dispatcher Version 8 Features to Drill Down to the Depths of Your Data. Thomas Kunselman. Western Users of SAS Software 2001

Contact Information

To request a copy of the ODS HTML forms tagsets please send e-mail requests to tkunselman@stan.csustan.edu.

Thomas Kunselman has spent over 15 years using many of the SAS system products. He has experience creating web-based reporting applications with varied types of data, including higher education, health care, pharmaceutical, and manufacturing process data.

Thomas Kunselman
Manager, Data Warehousing and Analytics
California State University, Stanislaus
801 West Monte Visa Avenue
Turlock, CA 95382
tkunselman@stan.csustan.edu
Voice: 209.667.3281
Fax: 209.667.3251
http://www.csustan.edu/Inst_Research