An Introduction to Creating and Using ODS Tagsets

Jack Hamilton, First Health, West Sacramento, California

ABSTRACT

ODS tagsets provide a greater level of control over the output created by the Output Delivery System than can be easily obtained by modifying existing templates. This paper gives an example of the use of tagsets in a SAS/Intrnet® application, but the ideas presented are applicable to non-web use as well.

INTRODUCTION

ODS, the Output Delivery System, is the mechanism recent versions of SAS Software use to give control of the appearance of SAS output to users. In version 6, we were, by and large, restricted to "line printer" output – monospace characters, no special effects such as bold or italic, no universal way to draw lines or boxes around your data, and certainly nothing fancy like special fonts or graphics.

In version 7, ODS was made available for most SAS procs which produce printed output, with increased support in version 8.

ODS ships with a number of destinations and styles. Loosely speaking, destinations describe where the ODS output will end up (HTML in a browser, PDF in a viewer, RTF in a word processor, and so forth), and styles describe what the output should look like (sansPrinter produces a fairly simple layout with a sans-serif font, BarrettsBlue has blue column and row headers, etc).

You can use PROC TEMPLATE to modify existing styles or create new ones. PROC TEMPLATE is documented in the online documentation, and Lauren Haworth's excellent book *Output Delivery System: The Basics* contains detailed documentation and a number of examples. You can use styles to modify your output in a number of ways, but if you've ever tried to do something radical, such as create a bare-bones HTML file without a lot of useless junk in it, or to insert extensive HTML headers, you've probably felt "there must be a better way".

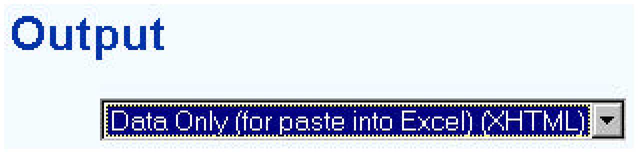
In version 8.2 (experimentally) and version 9 (production), the ability to completely define output, from scratch, was made available. The mechanism to do that, the ODS tagset, is that better way to create some types of output.

A tagset allows you to adopt a different, more algorithmic approach to creating output – instead of specifying an output destination and a style, and possibly a bazillion options on the ODS statement in your SAS procedure, you use an ODS tagset to tell SAS what to do when a particular kind of output is encountered (in tagset terms, this is an event).

MY APPLICATION

The application in which I first used tagsets was created using SAS/Intrnet. This app was intended to replace a Microsoft Access application; we had reached the limits of what it's practical to do in Access. We needed a development platform which could handle large sets of data (tens or hundreds of thousands of records), could do math, and had minimal distribution requirements. Although Access is good at math, it's not good at handling large numbers of records, and it's simply too delicate – anytime our systems folks upgraded Access or Windows, the application stopped working. SAS/Intrnet is more robust, and requires only a free browser on the user's end. The application also needed to be able to create output usable by Microsoft Excel.

There are several pages in the web app, and at the bottom of the last page this option appears:



The choices I wanted to have were

- *Data Only (for paste into Excel)*, which created unornamented XHTML suitable for import into recent versions of Excel;
- *Web Page (HTML)*, which created a more ornamented web page; and
- *Microsoft Excel (SYLK)*, which created a SYLK file suitable for import into older versions of Excel.

WHAT WAS WRONG WITH THE BUILT-IN SOLUTIONS?

You might ask, "Why not use the destinations built into SAS?" There were three reasons:

1. The HTML and XHTML created by SAS aren't very clean. A lot of extraneous junk is thrown in for no apparent good reason. It doesn't really hurt, other than making the resulting output much bigger than it needs to be, but it offends my sense of purity.
2. It's difficult to add custom headers using ODS. It can be done, but it's not pretty.
3. Although it might be possible to solve the two problems above using PROC TEMPLATE to create or modify styles, it still wouldn't be possible to create a SYLK file. That would require a data step.

USING TAGSETS

Using custom tagsets is easy. All you need is this HTML in the calling page:

```
<h2>Output</h2>

<select name="tagset">
   <option value="xhtml" selected>
      Data Only (for paste into Excel)
      (XHTML)</option>
   <option value="mtxhtml">Web Page
      (HTML)</option>
   <option value="sylk">
      Microsoft Excel (SYLK)</option>
</select>
```

And this code In the SAS program called by the SAS/Intrnet application:

```
ods path appread.tmplmst (read)
         sashelp.tmplmst (read);

ods markup file=_webout
           (title='RFP Counts Report')
            tagset=tagsets.&TAGSET.;
```

This assumes that the tagsets have already been created and stored in appread.tmplmst,

CREATING A TAGSET

There are at least two ways to create a tagset. You can modify an existing tagset, or you can start from scratch. I created the xhtml and mtxhtml tagsets by modifying existing tagsets. After I had finished, I knew enough about tagsets to create the sylk tagset from scratch (with help from some tools provided by SAS).

GETTING STARTED

The first step is to create a sample data set, and SAS code which will exercise all the features you want to include. In my application, I already had the data, and had decided that PROC REPORT was the only reporting procedure needed. Here's a small program which produces a sample report:

```
data results;
   infile cards;
   informat zip3 $3.;
   input zip3 c_iwc c_gs;
cards;
403 123 456
958 7 89
;;;;

proc report data=results nocenter nowindows;
   column zip3 c_iwc c_gs;
   define zip3   / order 'ZIP3' ;
   define c_iwc
      / display left format=comma10.0 ;
   define c_gs
      / display left format=comma10.0 ;
run;
```

MODIFYING AN EXISTING TAGSET

The easiest way to create a new tagset is to copy an existing tagset and make needed changes. To get a list of existing tagsets, use the following code:

```
proc template;
   list tagsets;
run;
```

Part of the listing will look like this:

```
Listing of: SASHELP.TMPLMST
Path Filter is: Tagsets
Sort by: PATH/ASCENDING

Obs    Path                   Type
 1     Tagsets                Dir
 2     Tagsets.Chtml          Tagset
 8     Tagsets.Event_map      Tagset
 9     Tagsets.Graph          Tagset
10     Tagsets.Htmlcss        Tagset
13     Tagsets.Phtml          Tagset
```

PHTML sounds like it might do the job (because I know from the ODS documentation that PHTML produces what SAS thinks is "minimal" HTML), and I can run a small job to see what it produces:

```
ods markup tagset=tagsets.phtml
    body='c:\temp3\phtml.txt';

/* Same PROC REPORT code as above.  */

ods markup close;
```

The output file (part of it – the whole thing is too large to include) looks like this (with some blanks removed):

```
   <html>
   <!-- Generated by SAS Software -->
   <!-- Http://www.sas.com -->
   <head>
   <style>
   <!--
   #l {text-align: left }
   #c {text-align: center }
   #r {text-align: right }
   #d {text-align: "." }
   #lb {text-align:left; vertical-align:bottom}
   #cb {text-align:center;vertical-align:bottom}
[…]
   <table border="1" cellspacing="1"
   cellpadding="7" rules="GROUPS" frame="BOX">
   <colgroup>
   <col>
   <col>
   <col>
   </colgroup>
   <thead>
   <tr>
   <td id="c">ZIP3</td>
   <td id="c">c_iwc</td>
   <td id="c">c_gs</td>
   </tr>
   </thead>
   <tbody>
   <tr>
   <td>403</td>
   <td>       123</td>
   <td>       456</td>
   </tr>
   <tr>
[…]
```

A casual examination reveals that this file includes a lot of stuff that isn't needed (the complete file is 90 lines long).

To look at the source code for the phtml tagset, run this code:

```
proc template;
   source tagsets.phtml;
run;
```

You'll get 700 lines of output, some of which looks like this:

```
define event top_file;
   start:
      put HTMLDOCTYPE NL NL NL;
      put "<html>" NL;
      put
     "<!-- Generated by SAS Software -->" NL;
      put "<!-- Http://www.sas.com -->" NL;
   finish:
      put "</html>" NL;
end;
```

Many of those 700 lines can be eliminated from your custom tagset, but at this point you have to know a bit more about how tagsets work.

EVENTS

The most important concept in tagsets is the *event*. An event is what happens when ODS tries to write out a particular kind of data. There are dozens of different events which your tagset definition *might* have to handle, but probably fewer than a dozen which you will *actually* care about. The rest can just be ignored. Many events won't occur at all for a given document.

The event whose code is shown above, top_file, is one of the important ones. It occurs once, at the beginning of processing. The other most important events are doc_body, table, row, and data.

STATEMENTS

Statements are associated with events, and tell ODS what to do when an event is encountered. The most important statements for simple tagsets are put, putl, and putq, which write to the ODS output file. The ndent and xdent statements are used to control indentation in the output (not strictly necessary with HTML, but an aid to reading). An event can have a statement condition, which controls whether the statement will be executed; it's similar in purpose to an IF statement.

VARIABLES

Variables are symbolic references to values. In simple tagsets, variables will have been set by the ODS supervisor. There are dozens of variables which might be set automatically; they're listed in the tagset documentation (it is also possible to set variables dynamically, but I don't completely understand how to do this). This example uses a variable with a statement condition:

```
   define event top_title;
      put "<title>";
      put "SAS Output Frame"
        / if !exist( value );
      put VALUE;
       put "</title>" NL;
   end;
```

The top_title event is called when ODS wants to write out the title for a document. In this example, it will place the title text inside an HTML title element. If the title hasn't been explicitly (using the TITLE= option on the ODS statement), a default value will be provided.

CREATING YOUR TAGSET

To create your new tagset, just copy the source for an existing tagset and delete what you think you don't want. After each change, make sure to run your sample job. It's easy at first to delete something you really need. After some experimentation, you'll get a feel for how tagset definitions work.

THE XHTML TAGSET

Here's the complete code for custom XHTML tagset. It was created using the iterative approach described above (I did add some additional ndent and xdent statements, and I further simplified some of the code):

```
   ods path appwrite.tmplmst (update)
            sashelp.tmplmst  (read);
   run;
   proc template;
      define tagset tagsets.xhtml;
         indent = 3;
         output_type = 'xml';
         split = '<br />';
         map = '<>&"';
         mapsub = '/&lt;/&gt;/&amp;/&quot;/';

         define event doc_meta;
         end;

         define event doc_title;
            put  '<title>';
            put  VALUE;
            put  '</title>' nl;
         end;
```

```
define event doc_head;
   start:
      put  '<head>' nl;
      ndent;
   finish:
      xdent;
      put  '</head>' nl;
end;

define event doc_body;
   start:
      put  '<body>' nl;
   finish:
      put  '</body>' nl;
end;

define event table;
   start:
      put  '<table border="1">' nl;
      ndent;
   finish:
      xdent;
      put  '</table>' nl;
end;

define event row;
   start:
      put  '<tr>' nl;
      ndent;
   finish:
      xdent;
      put  '</tr>' nl;
end;

define event table_head;
   start:
      put  '<thead>' nl;
      ndent;
   finish:
      xdent;
      put  '</thead>' nl;
end;

define event table_body;
   start:
      put  '<tbody>' nl;
      ndent;
   finish:
      xdent;
      put  '</tbody>' nl;
end;

define event table_foot;
   start:
      put  '<tfoot>' nl;
      ndent;
   finish:
      xdent;
      put  '</tfoot>' nl;
end;

define event data;
   start:
      put  "<th>"
        / if  cmp(section, "head");
      put  "<td>"
        / if !cmp(section, "head");
      put  VALUE;
   finish:
      put  "</th>" nl
        / if  cmp(section, "head");
```

```
            put  "</td>" nl
              / if !cmp(section, "head");
         end;

      end;

   run;
```

Here's the complete output resulting from using this tagset to display our sample data:

```
   <head>
      <title></title>
   </head>
   <body>
   <table border="1">
      <thead>
         <tr>
            <th>ZIP3</th>
            <th>c_iwc</th>
            <th>c_gs</th>
         </tr>
      </thead>
      <tbody>
         <tr>
            <td>403</td>
            <td>       123</td>
            <td>       456</td>
         </tr>
         <tr>
            <td>958</td>
            <td>         7</td>
            <td>        89</td>
         </tr>
      </tbody>
   </table>
   </body>
```

As you can see, this output is both easier to read and a third the size of the output created by the tagset supplied by SAS.

ADDING MORE CUSTOMIZATION

After you've gotten your first tagset working, you can add more custom features. I wanted an XHTML document header, an external CSS specification, and an external JavaScript file. I also wanted empty cells to contain an &nbsp; tag (so a cell border would be drawn), and I wanted all titles to contain a particular text string. I changed the top part of the definition above to be this:

```
   proc template;
      define tagset tagsets.mtxhtml;
         indent = 3;
         output_type = 'xml';
         split = '<br />';
         map = '<>&"';
         mapsub = '/&lt;/&gt;/&amp;/&quot;/';

         define event cell_is_empty;
            put %nrstr("&nbsp;");
         end;

         define event doc;
            start:
               put  '<?xml version="1.0"
   encoding="UTF-8"?>' nl;
               put  '<!DOCTYPE html PUBLIC "-
   //W3C//DTD XHTML 1.0 Transitional//EN"'
                    '
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   transitional.dtd">' nl;
               put  '<html
   xmlns="http://www.w3.org/1999/xhtml">' nl;
               ndent;
            finish:
               xdent;
               put  '</html>' nl;
         end;

         define event doc_meta;
            put '<meta http-equiv="pragma"
   content="no-cache">' nl;
            put '<style type="text/css">' nl
               '  @import
   "http://inside.firsthealth.com/providernetwor
   ks/prod/metrics/mtstyle-offset.css";' nl
               '  thead {font-weight: bold;}'
   nl
               '</style>' nl;
            put '<script language="JavaScript"
   type="text/javascript"' nl
               '
   src="http://inside.firsthealth.com/providerne
   tworks/prod/metrics/metrics.js"></script>'
   nl;
         end;

         define event doc_title;
            put  '<title>Metrics -';
            put  VALUE;
            put ' [Metrics - First Health]';
            put  '</title>' nl;
         end;

         define event doc_body;
            start:
               put  '<body>' nl;
               put  '<h1><script
   language="JavaScript"
   type="text/javascript">'
                    'mt_header("';
               put  TITLE;
               put  '");</script></h1>' nl;
            finish:
               put  '</body>' nl;
         end;

         define event table;
            start:
               put  '<table border="1">' nl;
               ndent;
            finish:
               xdent;
               put  '</table>' nl;
         end;
```

NON-MARKUP TAGSETS

Most output created using tagsets will be *markup*; a markup document contains data and information about the data, but not detailed instructions on how the data are to be displayed. HTML and XHTML, for example, may specify that a particular number is a cell in a table, and that the table has descriptive headers, and that the containing document has a title, but it's left up to the browser to figure out the details of how to display the cell, the table, and the document.

Tagsets can also be used to create non-markup documents. An example is CSV, which contains only data (and possibly column names), but no indication of what the columns might mean.

Another non-markup format, SYLK (for SYmbolic LinK), was created years ago as a way to transfer data between PC applications. Most spreadsheet programs since then have been able to read and write SYLK – even Microsoft Excel can do it, even though the SYLK specification is older than Excel and wasn't written at Microsoft.

Because SYLK is entirely text (no unprintable characters) and doesn't contain cryptic internal pointers, it's easy to write with a SAS program. I wanted to be able to import the results from my application into an Excel workbook with a minimum of effort. Recent versions of Excel can read HTML, but there was no guarantee that all users of my application would have a recent version. SAS doesn't provide a way to write an Excel workbook except on Windows, and the application runs under Unix. CSV files are straightforward, but a bit too bare-bones for my taste. I decided that writing a tagset to create SYLK would be the best compromise solution.

In this case, I had to start from scratch, as none of the existing tagsets created anything like SYLK.

Here's the tagset I ended up with:

```
   proc template;
      define tagset tagsets.sylk;
         indent = 0;
         map='"';
         mapsub='/""/';

         define event doc;
            start:
               put  'Content-type:
   application/vnd.ms-excel' nl;
               put  'Content-disposition:
   attachment;filename=rfpcount.sylk' nl nl;
               /* put  'Content-disposition:
   attachment;inline' nl nl; */
               put  'ID;PSAS8' nl;
               put  'P;F' nl;
               put  'P;F;SB' nl;
            finish:
               put  'E' nl;
         end;

         define event colspecs;
            start:
               put  'F;W1 ' colcount ' 20' nl;
         end;

         define event data;
            start:
               /* Header section (column headers
   are bold) */
               put  'F;SDM1;Y' row ';X' colstart
   nl /  cmp(section, "head");
               /* Always */
               put 'C;Y' row ';X' colstart;
               put ';K"';
               put value;
            finish:
               put '"';
               put nl;
         end;
      end;
   run;
```

COMING UP

Tagsets are experimental in SAS 8.2, and neither the syntax nor the implementation is quite as consistent as you might like. I found that the putl command didn't always work correctly. The events created by PROC REPORT weren't the same as those created by PROC PRINT for similar data. The system wasn't able to detect whether output was being written to a web page instead of a file (a useful capability which became available to templates in 8.0 or 8.1). There were other minor inconsistencies. Depending on what you want to do, it might be necessary to create several slightly different tagsets, perhaps one for each different output procedure or level of summarization.

I expect these problems to be solved in later releases. In addition, it will be possible to create graphics output.

OTHER POSSIBILITIES

CREATION BY INHERITANCE

Just as you can create a custom ODS template which inherits most of its properties from an existing template, you can create a custom tagset which inherits most of its properties from an existing tagset using the PARENT= attribute in the DEFINE TAGSET statement. This is helpful for putting together a tagset which is only slightly different, but in general, I think you might as well use the SOURCE statement to get the source of the original tagset and place that code directly into your definition. Everything is in one place, and if the definition of the base tagset is changed in a future version of SAS, your custom tagset won't mysteriously stop working.

USING EVENT_MAP

You might wonder what all the possible events for your data are, and what order they are called in. SAS has provided the EVENT_MAP tagset to help you figure these things out. All possible events and attributes are defined in EVENT_MAP, which might not be the case for the "real" tagsets. Create an event map like this:

```
   ods markup type=event_map
       body='c:\temp3\event.xml';
```

The resulting file is verbose, and can be hard to understand, but all the information you need is there. Here are the first 5 lines resulting from running our sample PROC REPORT:

```
   <doc operator="hamiltja" sasversion="8.2"
   saslongversion="8.02.02M0P01232001"
   date="2002-01-14" time="18:11:21"
   encoding="windows-1252" space="&nbsp;"
   split="<br>" event_name="doc"
   trigger_name="attr_out" state="start"
   class="Body" index="IDX" just="c">
     <doc_head event_name="doc_head"
   trigger_name="attr_out" state="start"
   class="Body" index="IDX" just="c">
       <doc_meta event_name="doc_meta"
   trigger_name="attr_out" state="start"
   empty="1" class="Body" index="IDX" just="c"/>
       <auth_oper event_name="auth_oper"
   trigger_name="attr_out" state="start"
   empty="1" class="Body" index="IDX" just="c"/>
       <doc_title event_name="doc_title"
   trigger_name="attr_out" state="start"
   empty="1" class="Body" index="IDX" just="c"/>
```

Another debugging tool is the text_map markup type, created using type=text_map. The output is much less verbose and may be easier for some people to read. Other options are described in the documentation.

REFERENCES

Documentation for tagsets in SAS 8.2 is available at

<http://www.sas.com/rnd/base/topics/odstagsets/>

Version 9 documentation will be shipped with the product, and will explain new features.

In addition to the ODS documentation provided as part of the SAS system, you might want to look at Lauren Haworth's *Output Delivery System: The Basics*, part of the SAS Books By Users series. Reliable rumor has it that other Books By Users on ODS topics are in the works.

CONTACT INFORMATION

Contact the author at:

Jack Hamilton
First Health
750 Riverpoint Drive
West Sacramento, California 95605
(916) 374-3833
jackhamilton@firsthealth.com