Paper TU03

Using tagsets to provide customised output: A tutorial

John Kirkpatrick, Insight Statistical Consulting Ltd, Huntingdon, Great Britain

ABSTRACT

The ODS is a flexible and reliable method of producing custom output to publication quality. However, using the conventional destinations (RTF, PDF and HTML) some effects and layouts are still difficult or even impossible to achieve. With tagsets, a much greater level of flexibility is achievable.

This tutorial will provide a basic introduction to the theory of tagsets and provide some practical guidance on how to go about developing your own tagsets. The example is based on moving `TITLE` and `FOOTNOTE` text from the document header to the body of a data table.

INTRODUCTION

The use of ODS styles and table templates provides SAS users with a powerful tool for providing customised output to their clients. However, there are some things that are beyond even styles and table templates. For example, the order in which output elements are created, and their positions, are not easily altered. In such situations, a tagset may well be able to help.

In this tutorial, I will introduce the basic principles behind the use of tagsets and, by way of a simple example, demonstrate how an existing template can be modified – or an entirely new tagset created – to do what you need. Although the example is based on an HTML tagset, the techniques used are applicable to all tagsets supplied by SAS Institute: or indeed to those that you write yourself. I don't claim that using a tagset is the best way to do what I do: it is simply a way, and the simplicity of the example allows me to concentrate on how the tagset is working. I assume that you are comfortable with using the `template` procedure to define custom styles or table templates. Some familiarity with HTML syntax would be helpful, but not essential.

The code in this tutorial has been tested using SAS® 9.1.3 on Windows XP Professional (Service Pack 2). The behaviour and attributes of tagsets has changed considerably between SAS version 8 and SAS 9, so the tutorial code is not really suitable for use with SAS version 8. However, the concepts described still hold true, and the corresponding version 8 code should not be too difficult to write.

PREPARING FOR THIS TUTORIAL

In what follows, I assume that the tagsets supplied by SAS Institute are already installed on your system. These should be installed by default for SAS 9, but you will need to install them yourself if you are using SAS Version 8. If you know that they are, you can safely skip this section and proceed to *Some Basic Concepts* below.

To check if the SAS Institute tagsets are installed, start SAS and submit the following code:

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

If the tagsets are installed, you'll see a long list appear in the log window. (We're particularly interested in `tagsets.event_map` and `tagsets.phtml`.) If not, you'll have to install the tagsets for yourself. They can be downloaded from the SAS Institute website (at the time of writing, the URL is http://support.sas.com/rnd/base/topics/odsmarkup/). Make sure you download the correct tagsets for your version of SAS. After downloading, installation is easy: simply extract and run the SAS program.

SOME BASIC CONCEPTS

A tagset is the set of rules that defines the SAS system's actions when creating output in a markup language. A markup language is defined to be "A set of labels that are embedded within text to distinguish individual elements or groups of elements for display or identification purposes. The labels are typically known as 'tags'." [1] The most familiar markup languages are probably HTML (the HyperText Markup Language) and XML (the eXtensible Markup Language). In HTML, the "tags" referred to in the definition include the `<b>` and `</b>` tags that define that the text between them should appear as bold.

When SAS writes output to a markup destination, it does so by responding to a series of *events* that are triggered by the ODS during the writing process. A tagset defines the actions that the ODS should carry out in response to each event. Thus, we might expect a tagset to find instructions about what do when a new document is created, when a new page is started, when a inline formatting command is encountered and so on. Indeed we do. As a specific example, we would expect an HTML-based tagset to contain a response to a "start of document" event that emits the <HTML> tag and a response to an "end of document" event that emits the </HTML> tag. As we will soon see, this is exactly the case.

The power of tagsets comes from the separation of event generation from event response. All SAS does is generate the event. The tagset defines the response. This means that the SAS system doesn't need to know anything about the syntax of the output file. As users can modify tagsets, or even create entirely new tagsets, this places a huge degree of flexibility in their hands.

INTRODUCING OUR EXAMPLE

In the rest of this tutorial, we'll modify the layout of the file produced by the following program:

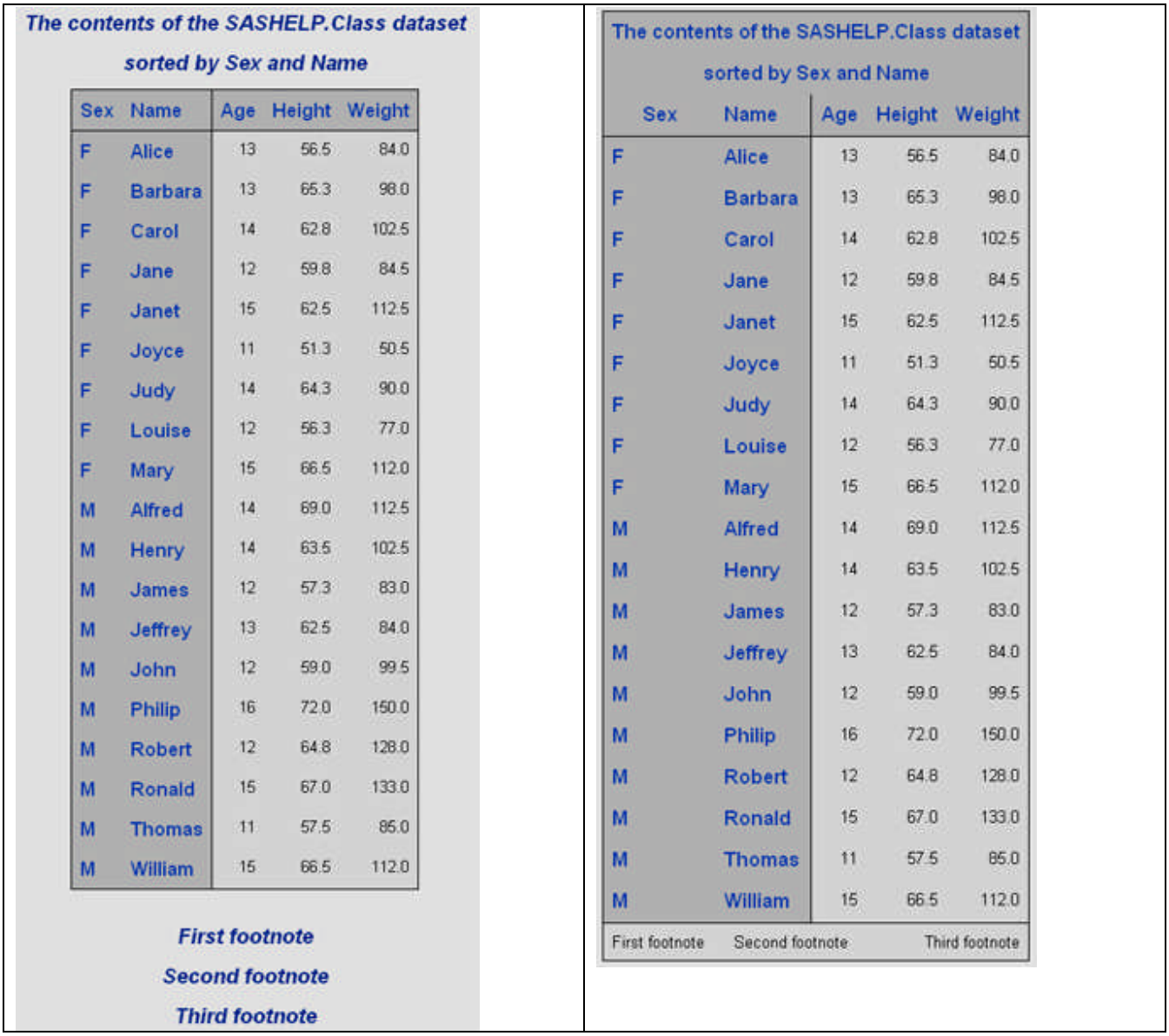
```
proc sort data=sashelp.class out=class;
  by sex name;
run;

ods listing close;
ods markup type=tagsets.phtml file="basic_output.htm";

proc print data=class;
  var age height weight;
  id sex name;
  by sex name;
  title1 "The contents of the SASHELP.Class dataset";
  title2 "sorted by Sex and Name";
  footnote1 "First footnote";
  footnote2 "Second footnote";
  footnote3 "Third footnote";
run;
ods markup close;
```

Listing 1: basic_output.sas

And we'll do it without modifying any of the code between the `proc print` and `run;` statements. In particular, we'll move the titles from the page header to the top of the table and the footnotes from three lines at the bottom of the page to a single line at the foot of the table. Here's a side-by-side comparison of the "before" and "after" listings.



The contents of the SASHELP.Class dataset

sorted by Sex and Name

Sex	Name	Age	Height	Weight
F	Alice	13	56.5	84.0
F	Barbara	13	65.3	98.0
F	Carol	14	62.8	102.5
F	Jane	12	59.8	84.5
F	Janet	15	62.5	112.5
F	Joyce	11	51.3	50.5
F	Judy	14	64.3	90.0
F	Louise	12	56.3	77.0
F	Mary	15	66.5	112.0
M	Alfred	14	69.0	112.5
M	Henry	14	63.5	102.5
M	James	12	57.3	83.0
M	Jeffrey	13	62.5	84.0
M	John	12	59.0	99.5
M	Philip	16	72.0	150.0
M	Robert	12	64.8	128.0
M	Ronald	15	67.0	133.0
M	Thomas	11	57.5	85.0
M	William	15	66.5	112.0

First footnote

Second footnote

Third footnote

The contents of the SASHELP.Class dataset

sorted by Sex and Name

Sex	Name	Age	Height	Weight
F	Alice	13	56.5	84.0
F	Barbara	13	65.3	98.0
F	Carol	14	62.8	102.5
F	Jane	12	59.8	84.5
F	Janet	15	62.5	112.5
F	Joyce	11	51.3	50.5
F	Judy	14	64.3	90.0
F	Louise	12	56.3	77.0
F	Mary	15	66.5	112.0
M	Alfred	14	69.0	112.5
M	Henry	14	63.5	102.5
M	James	12	57.3	83.0
M	Jeffrey	13	62.5	84.0
M	John	12	59.0	99.5
M	Philip	16	72.0	150.0
M	Robert	12	64.8	128.0
M	Ronald	15	67.0	133.0
M	Thomas	11	57.5	85.0
M	William	15	66.5	112.0

First footnote Second footnote Third footnote

Screen image 1: The original (left) and modified (right) output from the print procedure.

WATCHING EVENTS UNFOLD: THE EVENT_MAP TAGSET

Before looking at a tagset in detail, let's look at the events that occur whilst the output is actually created. This will enable us to put what we see in the tagset in context. The simplest way to watch events happen is to use the `tagsets.event_map` tagset. This tagset, rather than producing an output focussed on data, produces one based on events. The change to listing 1 is simple: change

```
ods markup type=tagsets.phtml file="basic_output.htm";
```

to

```
ods markup type=tagsets.event_map file="basic_event_map.txt";
```

(It's important to change the file type to something other than htm. Otherwise, when you open the file, your web browser will be the default editor and the file may *appear* to be empty.) Open the output file in a text editor.

Here's the first few lines of the file produced when the revised code is run:

```
<?xml version="1.0" encoding="windows-1252"?>

<doc operator="John" sasversion="9.1" saslongversion="9.01.01M3P07282004"
     date="2005-08-17" time="10:28:44" encoding="windows-1252" event_name="doc"
     trigger_name="attr_out" class="Body" index="IDX" just="c">
  <doc_head event_name="doc_head" trigger_name="attr_out" class="Body" index="IDX"
            just="c">
    <doc_meta event_name="doc_meta" trigger_name="attr_out" class="Body" index="IDX"
              just="c"/>
    <auth_oper event_name="auth_oper" trigger_name="attr_out" class="Body" index="IDX"
               just="c"/>
    <doc_title event_name="doc_title" trigger_name="attr_out" class="Body" index="IDX"
               just="c"/>
    <stylesheet_link event_name="stylesheet_link" trigger_name="attr_out" index="IDX"
                     just="c"/>
    <javascript event_name="javascript" trigger_name="attr_out" class="Body" index="IDX"
                just="c">
      <startup_function event_name="startup_function" trigger_name="attr_out"
                        class="StartUpFunction" index="IDX" just="c">
      </startup_function>
      <shutdown_function event_name="shutdown_function" trigger_name="attr_out"
                         class="ShutDownFunction" index="IDX" just="c">
      </shutdown_function>
    </javascript>
  </doc_head>
  <doc_body event_name="doc_body" trigger_name="attr_out" class="Body" index="IDX"
            just="c">
    <proc event_name="proc" trigger_name="attr_out" name="Print" index="IDX" just="c">
```

Output 2: a fragment from basic_event_map.txt

This file is best viewed in landscape orientation, when each line represents a single event. To help legibility, I have added some line breaks to the output fragment above. What we see is a series of events being triggered. The information relating to each event starts with an open angle bracket (<) and ends with a close angle bracket (>). The first piece of information about the event is its name, such as `doc`, `doc_head` and so forth. Within each event, there then follows a space delimited list of variable names and values in key=value format. For example, two of the variables associated with the `doc` event are `date` and `time`, with values of 2005-08-17 and 10:28:44 respectively. You'll be unsurprised to learn that I'm writing this part of the tutorial at about half past ten on the 17th of August 2005. Notice that not every variable is defined within every event.

Events can themselves trigger other events. This is indicated by the indentation in the output file. For example, in Output 2 above, the `doc_head` event triggers the `doc_meta`, `auth_oper`, `doc_title`, `stylesheet_link` and `javascript` events. The `javascript` event triggers the `startup_function` and `shutdown_function` events.

Having to define all these events in every single custom tagset we create is a daunting prospect. Fortunately, we don't have to. Two things help us. First of all, tagsets can inherit behaviour from other tagsets, just as styles and table templates can inherit from their ancestors. Just as usefully, if an event is redundant in a particular tagset, we can simply forget it: when SAS triggers an event that is undefined in the current tagset, it is simply ignored.

From the names of the events in this code fragment, it looks as if they carry out the actions necessary to create the header section of the output file. The last two lines of the fragment seem to be the start of the actual body of the document and the start of the output from a specific procedure. If this is correct, we would expect to see events related to document headers and footers following shortly after this. And we do:

```
<system_title_group event_name="system_title_group" trigger_name="attr_out" class="Body"
                    colcount="1" index="IDX" just="c">
  <title_container event_name="title_container" trigger_name="attr_out"
                   class="SysTitleAndFooterContainer" colcount="1" index="IDX" just="c">
    <title_container_specs event_name="title_container_specs" trigger_name="attr_out"
                           class="SysTitleAndFooterContainer" colcount="1" index="IDX"
                           just="c">
      <title_container_spec event_name="title_container_spec" trigger_name="attr_out"
                            colcount="1" type="string" index="IDX" just="c" width="100%"/>
    </title_container_specs>
```

```
    <title_container_row event_name="title_container_row" trigger_name="attr_out"
                         colcount="1" index="IDX" just="c">
      <system_title event_name="system_title" trigger_name="attr_out" class="SystemTitle"
                    value="The contents of the SASHELP.Class dataset" colcount="1"
                    index="IDX" just="c">
      </system_title>
    </title_container_row>
  </title_container>
</system_title_group>
```

Output 3: a fragment from basic_event_map.txt

So it would seem that our basic approach for moving the titles to the top of the output table should be focussed on these events.

In the next section, we'll examine the `tagsets.phtml` tagset code and start to make our first changes.

EXAMINING THE TAGSET CODE

The `TEMPATE` procedure code to view a tagset definition is simply

```
proc template;
  source <tagset name>;
run;
```

In interactive SAS, the code can be examined using the template browser. The simplest way to open the template browser is to type `odstemplates` in the SAS command window. Then expand the tree view in the left-hand pane and right-click on `phtml` in the right hand pane. Select `open` from the popup menu that appears. Do this for the `phtml` tagset. The structure of the code should look familiar: there's a `define tagset` statement block and, nested within it, a series of `define event` statement blocks. These blocks perform the obvious functions. Scroll down until you find the `define event proc` block. It looks like this:

```
define event proc;
   put "<div class=""branch"">" NL;
finish:
   put "</div>" NL;
end;
```

Source 2: A fragment from the tagsets.phtml tagset definition

This simple fragment illustrates several features of an event definition. The simplest feature is that a tagset writes to the output destination using the `put` statement. The tagset `put` statement acts in much the same way as the data step `put` statement: it writes either literal strings or variables to a file. The `NL` variable writes a new line character string to the output destination. So it looks as if the `proc` event writes two lines to the output destination, namely:

```
<div class="branch">
</div>
```

Is this the case? Well, not quite. We've ignored the `finish:` statement in the event definition. This is our first indication of an important new concept that, in terms of the ODS, is unique to tagsets: the elements contained in tagsets (that is, events), unlike the elements contained in styles (that is, style elements) or table templates (that is, column, footer and header elements) may have states.

There are two possible states: the *start state* and the *finish state*. Events can also be *stateless*. If an event is stateless, all the code between the define event statement and the corresponding `end;` is executed when the event is triggered. If an event is triggered in the start state, only the code between the `start:` label and finish: label is executed. (If there is a `finish:` label present, but no `start:` label, then an implicit `start:` label is assumed to occur immediately after the `define` statement. This is the case in *Source 2* above.) If an event is triggered in the finish state, only the code between the `finish:` label and the `end;` statement is executed. If no code corresponding to the state of the event exists, no error occurs.

So what actually happens here is that the tagset emits

```
<div class="branch">
```

when the `proc` event is called in the start state, and the tagset emits

```
</div>
```

when the `proc` event is called in the finish state.

Those of you familiar with object oriented programming might find it helpful to think of the start state as akin to an object constructor and the finish state to an object destructor.

When using the `tagsets.event_map` tagset, an event's start state is indicated by the <*[eventname]*> tag, and the finish state by the </*[eventname]*> tag.

MOVING THE TITLES

Returning to our example, the first thing we want to do is move the titles into the table header and format them correctly.

PLAYING POOH STICKS: FUN WITH STREAMS

To do this, we need to save the output created by the `system_title_group` event at the time it is triggered and save it until we need it when the table header is constructed. We need a stream to do this.

In tagsets, a stream can be used to create an output string over time, storing it in memory until it is actually needed. Only one stream can be open at any time. When a stream is open any output created by put statements is written to the stream rather than to the destination. Streams are opened with the `open` statement and closed with the `close` statement.

A quick check of the `tagsets.phtml` tagset shows that the `system_title_group` event is undefined, so we don't have to worry about overwriting any existing code. To begin with, we simply define our new tagset as

```
proc template;
  define tagset myHTML/store=work.tagsets;
    parent=tagsets.phtml;
    log_note="Using the myHTML tagset created at &systime on &sysdate9";
    define event system_title_group;
    start:
      open tstream;
    finish:
      close;
    end;
  end;
run;
```

Source 3: A fragment from the block_titles.sas program

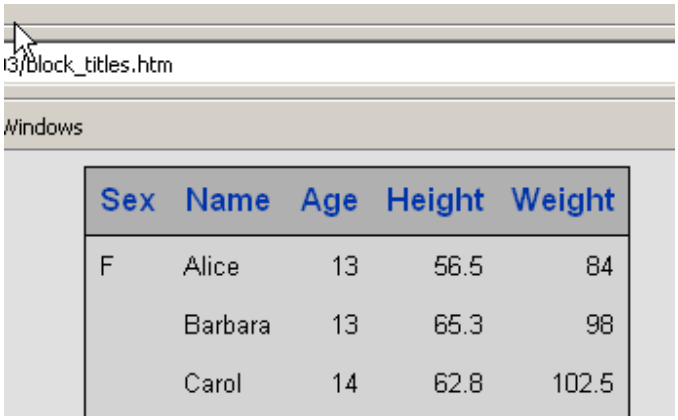
Note the use of the `parent` statement to inherit the behaviour of the `tagsets.phtml` tagset. The `log_note` statement simply adds a diagnostic message to the program log file to help us check that the correct tagset is being used.

Now simply adding an `ods path` statement and modifying the `ods markup` statement appropriately allows the program to run with the expected results.

```
ods path (prepend) work.tagsets;
ods markup type=myHTML file="block_titles.htm";
```

Source 4: Another fragment from the block_titles.sas program

As we're not actually using the `tstream` variable once we've created it, all this does is suppress the titles in the output file.



Screen image 2: The top of the block_titles.htm file, showing that the titles have been successfully suppressed

Now, the simplest thing to do would be to trigger the `system_title` event somewhere within the event that handles the start of the page output. Looking at `basic_event_map.txt` again, this is obviously `table_head` event. The code for the `table_head` event in the `tagsets.phtml` tagset is

```
define event table_head;
  put "<thead>" NL;
finish:
  put "</thead>" NL;
end;
```

Source 5: The definition of the table_head event in the tagsets.phtml tagset

So an obvious amendment is

```
define event table_head;
  put "<thead>" NL;
  putstream tstream;
finish:
  put "</thead>" NL;
end;
```

Source 6: The definition of the table_head event in the myHTML tagset in reinstate_titles1.sas

"DON'T CALL ME DOCTOR, I'M A SURGEON!": GETTING THE TITLES RIGHT

At first sight, when `reinstate_titles1.htm` is examined, the results are disappointing: it's identical to `basic_output.htm`. Our changes have had no effect. But the important thing is that we have successfully changed the *timing of the creation* of the titles. What we need to do now is change their *appearance* so that they blend seamlessly with the rest of the table. We need to find out which event in `tagsets.phtml` is responsible for the actual creation of the system titles. Looking at `basic_event_map.txt`, the obvious, and unsurprising, answer is that it's the `system_title` event: here's the relevant extract from `basic_event_map.txt`.

```
<title_container_row event_name="title_container_row" trigger_name="attr_out" colcount="1"
                     index="IDX" just="c">
   <system_title event_name="system_title" trigger_name="attr_out" class="SystemTitle"
                 value="The contents of the SASHELP.Class dataset" colcount="1"
                 index="IDX" just="c">
   </system_title>
</title_container_row>
<title_container_row event_name="title_container_row" trigger_name="attr_out" colcount="1"
                     index="IDX" just="c">
   <system_title event_name="system_title" trigger_name="attr_out" class="SystemTitle"
                 value="sorted by Sex and Name" colcount="1" index="IDX" just="c">
   </system_title>
</title_container_row>
```

Output 4: a fragment from basic_event_map.txt

And the corresponding event definition in `tagsets.phtml` is

```
define event system_title;
start:
  put "<h1";
  trigger align;
  put ">";
  put VALUE;
  break /if exists( value);
  put %nrstr(" ") /if exists( empty);
finish:
  put "</h1>" NL;
end;
```

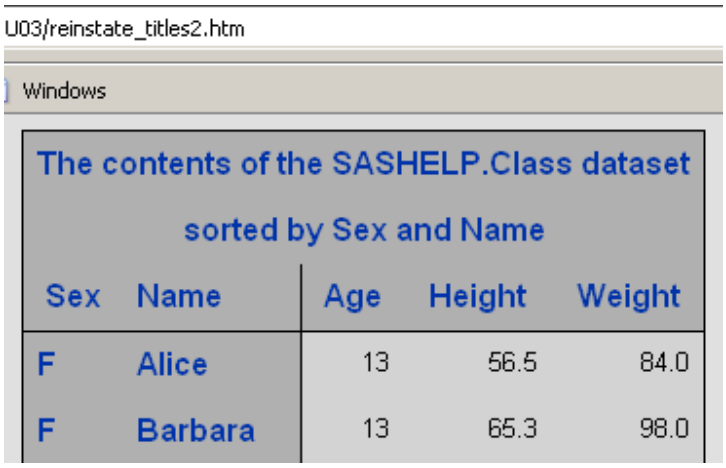
Source 7: The definition of the system_title event in tagsets.phtml

It should be pretty clear that the effect of this code is to wrap the title text in the HTML `<h1>` tag, which is what provides its format in the output file. So all we need to do is to modify the HTML wrapper to one more suitable for a table header. Here's some code that does the trick.

```
define event system_title;
start:
  put "<tr><th scope=colgroup colspan=5>" NL;
  put value NL;
  put "</th></tr>" NL;
end;
```

Source 8: The definition of the system_title event in the myHTML tagset in reinstate_titles2.sas

This produces the following output file:



Screen image 3: The top of the reinstate_titles2.htm file, showing that the titles have been successfully moved to the table header

As an aside, it's probably more correct to write

```
define event system_title;
  start:
    put "<tr><th scope=colgroup colspan=5>" NL;
    put value NL;
  finish:
    put "</th></tr>" NL;
end;
```

but in this simple case, there is no difference between the two options.

If you're *really* on the ball, you may have noticed a little bit of hard coding in the `system_title` event that could cause problems: I've assumed that the table will always have five columns. If it doesn't, the table header will look ugly. There are fixes to this problem, but they're quite involved, and deflect from the overall flow of this tutorial, so I shalln't fix it at the moment.

That's sorted the titles out. Now for the footnotes.

MOVING THE FOOTNOTES

The tasks I need to accomplish to get the footnotes looking the way I want are in many ways similar to what we had to do for the titles: we need to delay the termination of the table until after the footnote texts have been emitted and alter the formatting of the footnotes so that they look like they're part of the table. However, I'll use some different techniques, partly to show you some other features of tagsets, and partly because it's easier to do so. For reasons that will become clear, I'm going to reverse the order of the changes to the tagset this time round: I'll start with reformatting the footers, and then I'll move them into the table.

GIVING THE FOOTNOTES A FACELIFT

As we might expect, the event that creates the HTML that displays the footnote text is `system_footer`. Here's its definition in the `tagsets.phtml` tagset.

```
define event system_footer;
  start:
  put "<h1";
  trigger align;
  put ">";
  put VALUE;
finish:
  put "</h1>" NL;
end;
```

Source 9: The definition of the system_footer event in tagsets.phtml

It looks remarkably similar to the `system_title` event, doesn't it? What we need to do now is change the formatting from that of a header to that of a table cell. Here's the first step:

```
define event system_footer;
  start:
  put "<td>";
  put VALUE;
finish:
  put "</td>" NL;
end;
```

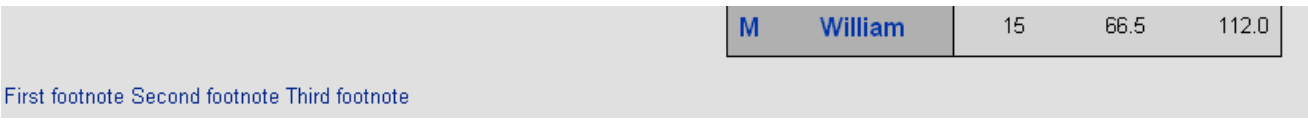
Source 10: The definition of the system_footer event in reformat_footnotes.sas

I want to put all three footnotes on the same line, so I can't put the `<tr>` tag inside the `system_footer` event: if I did, I'd still get three lines in the output file. I need to emit the `<tr>` before `system_footer` is triggered for the first time, and the `</tr>` immediately after it's triggered for the last time. Looking back at `basic_event_map.txt` once more, the `system_footer_group` is the obvious place to do this.

```
define event system_footer_group;
start:
  put "<tr>";
finish:
  put "</tr>";
end;
```

Source 11: The definition of the system_footer_group event in reformat_footnotes.sas

Running the revised code produces footnotes that look like this:



Screen image 4: The footnotes in the reformat_footnotes.htm file

The format is a long way from perfect, but at least I've got all three on the same line, which is all I wanted to do for now.

To get the footers looking exactly right, I need to roll my sleeves up and do a bit of investigation. When I open the reformat_footnotes.htm file in a text editor and scroll to the bottom, this is what I see. I've added comments to explain what everything is and where it comes from.

Contents of reformat_footnotes.htm	Description	Creating event
`<td class="r">112.0</td>`	Bottom right cell in the data table	`data`
`</tr>`	End of the final row in the data table	`row` (finish state)
`</tbody>`	End of the table body	`table_body` (finish_state)
`</table>`	End of the whole table	`table` (finish state)
`</div>`	End of table tag	
`</div>`	End of output tag	`output` (finish state)
` `	Line break	
`<tr>`	The start of a table row	`system_footer_group` (start state)
`<td>First footnote</td>`	The text of the first footnote	`system_footer`
`<td>Second footnote</td>`	The text of the second footnote	`system_footer`
`<td>Third footnote</td>`	The text of the third footnote	`system_footer`
`</tr>`	The end of a table row	`system_footer_group` (finish state)

The fact that there are so many events being triggered at this point in the program gives us a wide choice of approaches- not all of which will have the expected results! I could trigger the `system_footer_group` event manually at the end of the `table_body` event (or at the start of the table event) and then suppress it from triggering where it usually does, but generally speaking, I've found it better to delay events that interfere with what we're trying to do rather than to trigger the desired events early. Therefore, I'm going to take to opposite approach and use `table_body` event to prevent the table and output events from triggering until I want them to. This is straightforward:

```
define event table_body;
  put "<tbody>" NL;
finish:
  put "</tbody>" NL;
  block table;
  block output;
end;
```

Source 12: The definition of the table_body event in move_footnotes1.sas

The `block` statement prevents an event from being triggered, either automatically or manually, until it is `unblock`ed. So this modification of the `table_body` event will prevent the `<table>`, `</div>` and `</div>` tags from being emitted where they were previously. This clears the way for my revised `system_footer_group` event. But I must ensure that the table and output events do eventually get triggered: otherwise, the output file will be malformed.

```
define event system_footer_group;
start:
  put "<tr>" NL;
finish:
  put "</tr>" NL;
  unblock table;
  unblock output;
  trigger table finish;
  trigger output finish;
end;
```

Source 13: The first definition of the system_footer_group event in move_footnotes1.sas

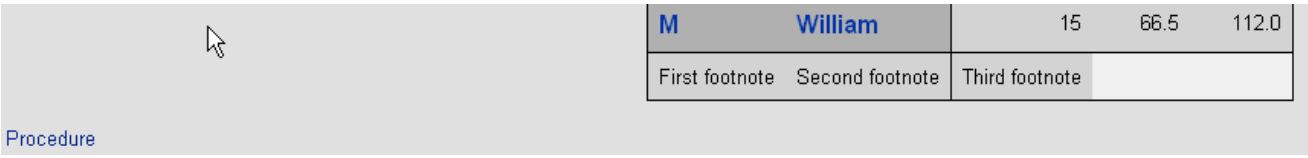
One *gotcha* with `block` and `unblock` statements is that an event must be `unblock`ed at least as many times as it was `block`ed before it can be triggered once more. This makes the placement of the `block` and `unblock` statements crucial.

Ideally, the footnotes should be wrapped in an HTML `<tfoot>` tag, just as the titles and column headers are wrapped in a `<thead>` text. Doing this will ensure that long tables display correctly in modern browsers. I could write an event to do this for me, but there is one in the `tagsets.phtml` tagset already: it's just not triggered by the `print` procedure in this example. So I'll simply use the pre-existing event.

```
define event system_footer_group;
start:
  trigger table_foot start;
  put "<tr>" NL;
finish:
  put "</tr>" NL;
  trigger table_foot finish;
  unblock table;
  unblock output;
  trigger table finish;
  trigger output finish;
end;
```

Source 14: The final definition of the system_footer_group event in move_footnotes1.sas

Running this code produces an output file with footnotes like this:



Screen image 5: The footnotes in the move_footnotes1.htm file

The footnotes are now within the main body of the table (admittedly they do still need some formatting), but I've produced a stray "Procedure" in the bottom left hand corner of the page. A little detective work revealed that this came from the `pre_post` event, which is triggered by the `doc_body` event in the finish state. I didn't investigate this too closely, but I suspect the reason is that by tampering with the order in which the various events are triggered, I've created an unexpected change in the environment that is assumed by `pre_post`. Fortunately, a fix was straightforward. Here's the original definition of the `pre_post` event from `tagsets.phtml`.

```
define event pre_post;
start:
  put PREHTML;
  putq "<img alt="""" src=" PREIMAGE;
  put $empty_tag_suffix /if exists( PREIMAGE);
  put ">" /if exists( PREIMAGE);
  put PRETEXT;
finish:
  put POSTTEXT;
  putq "<img alt="""" src=" POSTIMAGE;
  put $empty_tag_suffix /if exists( POSTIMAGE);
  put ">" /if exists( POSTIMAGE);
  put POSTHTML;
end;
```

Source 15: The pre_post event in tagsets.phtml

Removing the highlighted line fixed the problem.

All that now remains to be done is to fix the last few gremlins with the footnotes. The untidy appearance of the body table is caused by the fact that I've defined only three cells (one for each footnote) in the final row of the table, but there are five columns of data in the table body. Thus the rightmost two cells in the final row are undefined. The fix I've chosen is to span each of the second and third footnotes over two data columns. As all three footnotes are emitted by the same event, this gives me the chance to use a user defined tagset variable as well as some tagset conditional logic.

None of the automatic variables associated with the `system_footer` event indicate which footer is being emitted. I need to create an index to count which footer is currently being dealt with.

```
define event system_footer_group;
start:
  trigger table_foot start;
  put "<tr>" NL;
  eval $jk_index 0;
finish:
  put "</tr>" NL;
  trigger table_foot finish;
  unblock table;
  unblock output;
  trigger table finish;
  trigger output finish;
end;
```

Source 16: Defining an index variable in the system_footer_group event in final_format.sas

This creates my counter variable. The scope of tagset variables is the event in which they were created. In the same way as macro variables are available to macros called by the macro in which the variable was created, tagset variables are available to events triggered by the event in which the variables were created. No to perorm the count.

```
define event system_footer;
start:
  eval $jk_index $jk_index + 1;
  put "<td>";
  put VALUE;
finish:
  put "</td>" NL;
end;
```

Source 17: Incrementing the index variable in the system_footer event in final_format.sas

And finally to do something useful with it.

```
define event system_footer;
start:
  eval $jk_index $jk_index + 1;
  put "<td";
  put " colspan=2" /if $jk_index ge 2;
  put ">";
  put VALUE;
finish:
  put "</td>" NL;
end;
```

Source 18: Conditional execution in the system_footer event in final_format.sas

The syntax is somewhat awkward, but the meaning should be clear. As a final tweak, I'll centre align the second footnote and right align the third as follows:

```
define event system_footer;
start:
  eval $jk_index $jk_index + 1;
  put "<td";
  put " colspan=2" /if $jk_index ge 2;
  put " align='center'" /if $jk_index eq 2;
  put " align='right'" /if $jk_index eq 3;
  put ">";
  put VALUE;
finish:
  put "</td>" NL;
end;
```

Source 19: More conditional execution in the system_footer event in final_format.sas

This gives me the final version of the footnotes:

M	William	15	66.5	112.0
First footnote	Second footnote			Third footnote

Screen image 6: The footnotes in the final_format.htm file

Here's the full listing of the final version of the program:

```
proc sort data=sashelp.class out=class;
  by sex name;
run;

proc template;
  define tagset myHTML/store=work.tagsets;
    parent=tagsets.phtml;
    log_note="Using the myHTML tagset created at &systime on &sysdate9";
    define event system_title_group;
    start:
      open tstream;
    finish:
      close;
    end;
    define event system_title;
    start:
     put "<tr><th scope=colgroup colspan=5>" NL;
      put value NL;
      put "</th></tr>" NL;
```

```
    end;
    define event table_head;
      put "<thead>" NL;
      putstream tstream;
    finish:
      put "</thead>" NL;
    end;
    define event system_footer;
    start:
      eval $jk_index $jk_index + 1;
      put "<td";
      put " colspan=2" /if $jk_index ge 2;
      put " align='center'" /if $jk_index eq 2;
      put " align='right'" /if $jk_index eq 3;
      put ">";
      put VALUE;
    finish:
      put "</td>" NL;
    end;
    define event system_footer_group;
    start:
      trigger table_foot start;
      put "<tr>" NL;
      eval $jk_index 0;
    finish:
      put "</tr>" NL;
      trigger table_foot finish;
      unblock table;
      unblock output;
      trigger table finish;
      trigger output finish;
    end;
    define event table_body;
      put "<tbody>" NL;
    finish:
      put "</tbody>" NL;
      block table;
      block output;
    end;
    define event pre_post;
    start:
      put PREHTML;
      putq "<img alt="""" src=" PREIMAGE;
      put $empty_tag_suffix /if exists( PREIMAGE);
      put ">" /if exists( PREIMAGE);
      put PRETEXT;
    finish:
      putq "<img alt="""" src=" POSTIMAGE;
      put $empty_tag_suffix /if exists( POSTIMAGE);
      put ">" /if exists( POSTIMAGE);
      put POSTHTML;
    end;
  end;
run;

ods path (prepend) work.tagsets;
ods markup type=myHTML file="final_format.htm";

title1 "The contents of the SASHELP.Class dataset";
title2 "sorted by Sex and Name";
footnote1 "First footnote";
footnote2 "Second footnote";
footnote3 "Third footnote";
```

```
proc print data=class;
  var age height weight;
  id sex name;
  by sex name;
run;
ods markup close;
```

Source 20: final_format.sas

CONCLUSION

This tutorial has barely scratched the surface of what tagsets can do, but it has, I hope, shown you a practical approach to creating a tagset-based solution to problems. There is no doubt that tagsets are powerful, flexible and therefore complicated. It is likely that most day-to-day problems will have simpler solutions based on other approaches. But when all else fails, tagsets are worth a look.

The ODS learning curve is generally regarded as a steep one, and in my opinion, tagsets represent the steepest part of the mountain, but they can be worthwhile. Learning about tagsets is made more difficult by the fact that the documentation that ships with the standard SAS installation is out of date. The most useful reference I've found is on the SAS Institute website at http://support.sas.com/rnd/base/topics/odsmarkup/tagsets.html, which includes up-to-date dociumentation of all tagset statements, examples of their use, and the version of SAS in which they were first available.

REFERENCES

1. Computer Desktop Encyclopedia. Computer Language Company Inc., 2005. *Answers.com* GuruNet Corp. June 28, 2005. http://www.answers.com/topic/markup-language

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

John Kirkpatrick
Insight Statistical Consulting Ltd
Tile Barn
High Haden Road
Glatton
HUNTINGDON
Cambridgeshire PE 28 5RU
Great Britain

Work Phone: +44 (0) 1487 830838
Email: phuse2005@isc-ltd.co.uk (Please note: I get a lot of spam email. To maximise my chances of reading your email, please use an informative header and avoid free email accounts such as MSN and hotmail.)
Web: www.isc-ltd.co.uk