

Inline Formatting with ODS Markup

Eric Gebhart, SAS Institute, Cary, NC.

ABSTRACT

Explore the power of inline formatting in ODS. And, learn about the super powers of inline formatting with ODS markup and ODS measured; see how easy it is to make your reports look better than ever before.

With the new and improved syntax, ODS Inline formatting is better and more powerful than ever before. Learn how to use it and extend it to do even more than you dreamed possible.

TWAS BRILLIG AND THE SLITHY TOVES, DID GYRE AND GIMBLE IN THE WABE

Inline formatting has been around for a few years now, but as of 9.2 it has a new syntax and is more powerful than ever.

However, it can look a bit daunting and much like the feelings you may have about Lewis Carol's Jabberwock and the Jub Jub bird and frumious Bandersnatch. You might be just as well off running off the other way. But inline formatting isn't nearly so bad as the Jabberwock, and the Jabberwock didn't fair so well anyway.

'HE LEFT IT DEAD, AND WITH ITS HEAD, HE WENT GALUMPHING BACK.'

This paper will explain the new syntax and the format functions available in most ODS destinations. It will also explore how those functions work within the ODS Markup destinations. Putting that knowledge to good use, will culminate in creating new format functions to do more powerful manipulation of any ODS Markup destination. This knowledge will give new insight into how ODS Markup destinations can be manipulated and used in new ways. And not least, inline formatting will no longer be as scary as a fairy tale.

HE TOOK HIS VORPAL SWORD IN HAND:

LONG TIME THE MANXOME FOE HE SOUGHT --

The first step is to set the ODS Escape character to something that won't interfere with other text in your SAS program. Favorite characters are ~ and ^. There is also another way, (*ESC*), but a single escape character is easier to type and read. The ODS Escapechar statement is used to set the new escape character.

```
ODS Escapechar='~';
```

The rest of the syntax is fairly simple, The inline format begins right after the escape character with an open brace, {. The first thing inside the brace is the format function's name, and after that, the arguments for the function, followed by a closing brace, }.

Here is a simple example which uses an inline format function in a title.

```
Title 'This is ^{style [color=red] Red}';
```



Style is one of the most powerful built in format functions, it has the same style over-ride capabilities that have been available in Proc Tabulate, Report, and Print for years. The usual style caveat's from one destination to another still apply. Among the more useful style attributes are color, font_style, font_family, font_weight, and text_decoration,

which includes, strike through, underline, overline and blink. Your mileage may vary depending on the destination. Background

can work, but again, it depends upon the destination. Some can do it, others can't.

In addition to Style there are quite a few other functions available for most destinations.

```
~{dagger}  
~{sigma}  
~{unicode <Hex|name>}  
~{super text}  
~{sub text}  
~{raw type text}  
~{style <style><[attributes]>}  
~{nbspspace count}  
~{newline count}
```

Dagger and Sigma are old functions that are all but replaced by the new Unicode function. The Unicode function can print any special character, Many of them by name, and if not by name, then by hex value. Super and Sub have been around for awhile, and they superscript and subscript text as you might have already guessed. The Raw function

is particularly useful for RTF and possible other markup languages for quick and easy insertion of raw markup code into the output file. The first argument to raw, if provided is the destination type that it is intended for, RTF, HTML, LaTeX, that is what keeps a raw inline format directed to it's proper destination. It is not as useful as it once was due to the way format functions work with ODS Markup, which includes the new tagsets.RTF destination. There are now better ways to do this. Style is probably the single most useful and powerful format function. Nbsp space will insert a non-breaking space for those destinations that support such a thing, if there is a count it will insert that many non-breaking spaces. The newline function will insert a newline, if there is a count it will insert that number of newlines.

One last thing before we move on, just a little thing, the new syntax for format functions allows for nesting. The output of one function can be the input to another. This allows for scoping of style, and simpler coding of complexformatting.

ONE, TWO! ONE, TWO! AND THROUGH AND THROUGH THE VORPAL BLADE WENT SNICKER-SNACK!

Here is an example of using most of these functions.

```
title "Examples of Functions";  
  
Title2 'This is ^{style [color=red] Red}';  
  
title3 'Example of ^{nbspspace 3} Non-Breaking Spaces Function';  
  
title4 'Example of ^{newline 2} Newline Function';  
  
title5 'Example of ^{raw \cfl2 RAW} RAW function';  
  
title6 'Example of ^{unicode 03B1} UNICODE function';  
  
title7 "Example ^{style [foreground=red] of ^{super ^{unicode ALPHA} ^{style  
[foreground=green] Nested}} Formatting} and Scoping";  
title8 "Example of ^{super ^{style [foreground=red] red ^{style [foreground=green]  
green } and ^{style [foreground=blue] blue}}} formatting";
```

Examples of Functions
This is Red
Example of Non-Breaking Spaces Function
Example of
Newline Function
Example of \cf12 RAW RAW function
Example of a UNICODE function
Example of a Nested Formatting and Scoping
Example of red green and blue formatting

Underlining works everywhere, but strike through and overlines are not as consistent. Not all target output types support all the things that can be done with ODS styles.

```
options nonumber nodate;
ods html file="underline.html";

ods escapechar='^';
title underlin=1 "an underlined title using underlin=";
title3 "^{style [textdecoration=line_through]a line-through title}";
title5 "^{style [textdecoration=overline]An overlined title}";
title7 "^{style [textdecoration=underline]Switching from underline to}
        ^{style [textdecoration=line_through]line-through, then}
        ^{style [textdecoration=overline]overline}.";

proc print data=sashelp.class(obs=1);run;

ods pdf text="^{style [just=r textdecoration=underline color=red]Here is some
random underlined text.}";

ods _all_ close;
```

an underlined title using underlin=.

~~a line-through title~~

An overlined title

Switching from underline to line-through, then overline.

The Unicode function is one of the most sought after functions, here is an example that shows how inline formatting can

be embedded in data.

```
ods escapechar='^';

/* Create a table of unicode characters */
data work.unicode;
input @1 name $25. @27 value $4.;
datalines;
Snowman                2603
Black Knight           265E
White Rook             2656
Snowflake              2744
Two Fifths             2156
Greater Than or Equal To 2267
;

/* Create table that will show the name, unicode value, and actual symbol */
proc template;
define table unitable;
  define column name;
    header = 'Name';
  end;
  define column value;
    style={textalign=center};
    header = 'Value';
  end;
  define column symbol;
    style={textalign=center};
    header = 'Symbol';
    compute as '^{'unicode ' || value || '}';
  end;
end;
run;

/* Make the fonts big */
proc template;
define style styles.bigprinter; parent=styles.printer;
  class systemtitle, data, header /
    fontsize = 40pt
  ;
end;
run;

/* Generate report */
ods pdf file="unicode.pdf" style=styles.bigprinter;
ods html file="unicode.html" style=styles.bigprinter;

data _null_;
  set work.unicode;
  file print ods=(template='unitable');
  put _ods_;
run;

ods _all_ close;
```

Name	Value	Symbol
Snowman	2603	
Black Knight	265E	
White Rook	2656	
Snowflake	2744	
Two Fifths	2156	$\frac{2}{5}$
Greater Than or Equal To	2267	$\geq$

BEWARE THE JABBERWOCK, MY SON!

THE JAWS THAT BITE, THE CLAWS THAT CATCH!

One of the problems with style, is that not every target output type is capable of every nuance of what ODS styles support.

ODS styles are mostly a superset of what all the destinations are capable of. A prime example is Excel. There are only

a few style attributes that excel can actually do. This is actually a limitation of Microsoft's SpreadsheetML specification.

Excel can do color, font_face, font_weight bold, and underline, but not overline, strikethrough or overline.

SO RESTED HE BY THE TUMTUM TREE,

AND STOOD AWHILE IN THOUGHT.

What is it that is so special about ODS Markup? It's the tagsets. More importantly it's how format functions are defined in

ODS markup destinations. They are defined as events in the tagset. This realization may be enough to make your head spin.

If it isn't, then let's see if we can get it going. Starting simple, and using the HTML destination as an example, lets see

what the format functions look like, knowing a bit of HTML is helpful here.

Here is the tagset code for Dagger.

```
define event dagger;
  start:
    put '&dagger;' ;
end ;
```

Here is the tagset code for super and sub.

```
define event super;
  start:
    put "<sup>" ;
    put VALUE;
    put "</sup>"
end ;

define event sub;
  start:
    put "<sub>" ;
    put VALUE;
    put "</sub>" ;
end ;
```

Those are a little more complicated, they have text that they need to handle, but nothing too bad.

The nbspace and newline functions are nearly identical, and quite a bit more complicated, if you can get past the tagset syntax you'll see good old data step functions doing data step like things. First convert the value to a number, make sure it's no bigger than 256 and no less than 0, then loop for that many times printing ' ace' for a non breaking space or in the case of newline, print
 for a break line.

```
define event nbspace;
  do /if value;
    break /if index(value,"-");
    eval $ncount inputn(value, "3.");
    do /if $ncount > 256;
      eval $ncount 256;
    done;
  else;
    eval $ncount 1;
  done;

  do /while $ncount;
    put '&nbspace;' ;
    eval $ncount $ncount-1;
  done;
  unset $ncount;
end;
```

Now, the question is, if format functions are just tagset events, and tagsets are choc full of events, what other events are there that might be useable as format functions? The short answer is almost any event. Some make sense, some don't, but you can always make one that does. You could use inline formatting to create an entire table with rows and cells, but I wouldn't recommend it, there are easier ways to do that.

Just a short look around in the html tagset reveals two functions that could be useful. Pagebreak and line. Pagebreak prints the contents of the style attribute, pagebreakHTML, usually a line of some sort, and the line event prints a simple <hr> line. Let's see what happens if we try that...

```
title1 "Example ^{style [foreground=red] of ^{line} a unicode Alpha ^{unicode ALPHA} ^{newline} ^{line}"
```

Example of

a unicode Alpha α with some lines

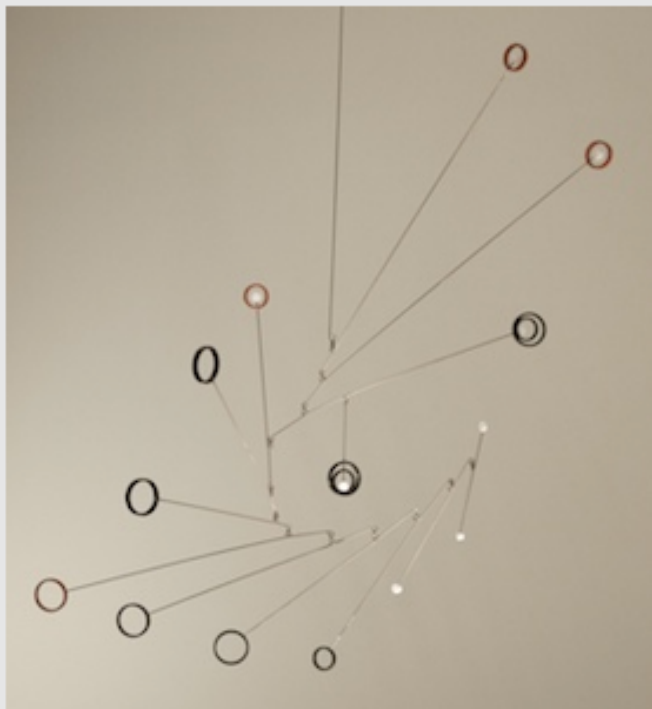
If that works, what else can we do? Lots! How about an image function?

```
define event inline_image;  
  putq "<img" "src=" value ">";  
end;
```

```
title2 "Example ^{style [color=green] of ^{line} an image {inline_image  
Droplet.jpg} ^{newline} ^{line}}"
```

Example of

an image



Droplet, a mobile I
made January 2009.

We've seen super and sub, they look like this.

```
title2 'This is ^{super SUPER text}  
      ^{sub This is SUB text}  
      ^{super More SUPER text}  
      ^{sub More SUB text}';
```

This is SUPER text This is SUB text More SUPER text More SUB text

How about a super duper function? Something that draws a box around the text and then superscripts and subscripts each word

alternately! Silly I know. But it can be done. Here's how.

```
proc template;
  define tagset tagsets.sdhtml;
    parent=tagsets.html4;

    define event superDuper;

      put '<span style="border-width:1; border-style:solid; border-color:
black;"> ' nl;

      eval $count 1;
      do /if index(value, ' ');
        set $word scan(value, 1, ' ');
        do /while !cmp($word, ' ');

          do /if mod($count, 2);
            put '<sup>' $word '</sup>';
          else;
            put '<sub>' $word '</sub>';
          done;

          eval $count $count + 1;
          set $word scan(value, $count, ' ');
        done;

      else /if $event;
        put '<sup>' value '</sup>';
      done;

      put nl "</span>" nl;

    end;
  end;
run;

ods escapechar="^";

title 'This is ^{superduper SUPER DUPER text}
      This is ^{sub SUB text}';

title2 'This is ^{super SUPER text}
        ^{sub This is SUB text}
        ^{super More SUPER text}
        ^{sub More SUB text}';

ods tagsets.sdhtml file="superduper.html";

proc print data=sashelp.class;
run;

ods _all_ close;
```

This is SUPER DUPER^{text} This is SUB text

**AND, AS IN UFFISH THOUGHT HE STOOD,
THE JABBERWOCK, WITH EYES OF FLAME,
CAME WHIFFLING THROUGH THE TULGEY WOOD,
AND BURBLED AS IT CAME!**

That does seem like a frivolous use of the new super powers of inline format functions and tagsets, but it does show that almost anything can be done. What else can we do? Well, I'm still coming up with idea's for that, and more frequently than not, necessity is the mother of invention. Here are a few things that have come up. Embedding images, Blocking and unblocking events to change the tagset's behavior. Setting and unsetting options on a tagset to change it's behavior. There isn't any rule that says an inline format function has to create output. One of the best and most useful things to do with an inline format function is to read external files of text or markup to be placed into the output.

Frequently, especially for HTML, it is desirable to put HTML code that is created and maintained by the web site police, into certain places within the report to make the output conform to certain guidelines about look and feel. That is easy to do with a inline format function. That same function could be quite convenient for placing disclaimers, and paragraphs of text that are most easily maintained elsewhere. Another purpose for this functionality is to include Flash code into the html file. That is something that is becoming more and more common.

There is already tagset code available that knows how to read a file. It is actually just a tagset version of the SAS manual page on fget. There is a bit of cleverness here that makes the readfile function a little more manageable. The process_data event is used to actually print the lines the readfile function reads. That makes it possible to leave the readfile event alone, and still repurpose it for other uses by modifying the process_data event. The last thing we need is an actual function to use as an inline format function. That is the include_file event.

```
define event readfile;

/*-----eric-*/
/*-- Set up the file and open it.                --*/
/*-----13Jun03-*/

set $filrf "myfile";
eval $rc filename($filrf, $read_file);

/*
do /if $debug_level >= 5;
    putlog "File Name" ":" $rc " : " $read_file;
done;
*/

eval $fid fopen($filrf);

/*
do /if $debug_level >= 5;
    putlog "File ID" ":" $fid;
done;
*/

/*-----eric-*/
/*-- datastep functions will bind directly to the --*/
/*-- variable space as it exists.                --*/
/*--                                              --*/
```

```

/*-- Tagset variables are not like datastep          --*/
/*-- variables but we can create a big one full     --*/
/*-- of spaces and let the functions write to it.    --*/
/*--                                                --*/
/*-- This creates a variable that is 200 spaces so  --*/
/*-- that the function can write directly to the    --*/
/*-- memory location held by the variable.          --*/
/*-- in VI, 200i<space>                             --*/
/*-----27Jun03--*/
set $file_record "

";

/*-----eric--*/
/*-- Loop over the records in the file              --*/
/*-----13Jun03--*/
do /if $fid > 0 ;

    do /while fread($fid) = 0;

        set $rc fget($fid,$file_record ,200);

        /*
        do /if $debug_level >= 5;
            putlog 'Fget' ':' $rc 'Record' ':' $file_record;
        done;
        */

        set $record trim($file_record);

        trigger process_data;

        /* trimn to get rid of the spaces at the end. */
        /*put trimn($file_record ) nl;*/

    done;
done;

/*-----eric--*/
/*-- close up the file.  set works fine for this.   --*/
/*-----13Jun03--*/

set $rc close($fid);
set $rc filename($filrf);

end;

define event process_data;
    put $record;
end;

define event include_file;
    set $read_file value;
    trigger readfile;
end;

```

As an example, lets read the poem, Jabberwocky, into our html output.

```

ods html file="jabberwocky.html";

ods text="~{style [font_style=italic] ~{include_file jabberwocky.txt}}";

ods html close;

```

'Twas brillig, and the slithy toves Did gyre and gimble in the wabe: All mimsy were the borogoves, And the mome raths outgrabe. "Beware the Jabberwock, my son! The jaws that bite, the claws that catch! Beware the Jubjub bird, and shun The frumious Bandersnatch!" He took his vorpal sword in hand: Long time the manxome foe he sought --So rested he by the Tumtum tree, And stood awhile in thought. And, as in uffish thought he stood, The Jabberwock, with eyes of flame, Came whiffing through the tulgey wood, And burbled as it came! One, two! One, two! And through and through The vorpal blade went snicker-snack! He left it dead, and with its head He went galumphing back. "And, has thou slain the Jabberwock? Come to my arms, my beamish boy! O frabjous day! Callooh! Callay!" He chortled in his joy. 'Twas brillig, and the slithy toves Did gyre and gimble in the wabe; All mimsy were the borogoves, And the mome raths outgrabe.

Ok, close but not quite. I really did want a newline between each line that was read. But that is only for this example. What if I really wanted those lines to run together? Well, we could create another inline function to change the behavior of `process_data`.

```
define event include_newlines;
  do /if cmp (value, 'True');
    set $include_newlines='True';
  else;
    unset $include_newlines;
  done;
end;
```

`Process_data` can now be modified to include newlines or not.

```
define event process_data;
  put $record;
  put "<br>" /if $include_newlines;
end;
```

Using the new function to change the behavior of the include file format function looks like this.

```
ods html file="jabberwocky.html";

ods text="~{style [font_style=italic] ~{include_newlines True} ~{include_file
jabberwocky.txt}}";

ods html close;
```

That's more like it. Now we have some nice HTML.

Jabberwocky, By Lewis Carroll.

*`Twas brillig, and the slithy toves
Did gyre and gimble in the wabe:
All mimsy were the borogoves,
And the mome raths outgrabe.*

*"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"*

*He took his vorpal sword in hand:
Long time the manxome foe he sought --
So rested he by the Tumtum tree,
And stood awhile in thought.*

*And, as in uffish thought he stood,
The Jabberwock, with eyes of flame,
Came whiffling through the tulgey wood,
And burbled as it came!*

*One, two! One, two! And through and through
The vorpal blade went snicker-snack!
He left it dead, and with its head
He went galumphing back.*

*"And, has thou slain the Jabberwock?
Come to my arms, my beamish boy!
O frabjous day! Callooh! Callay!"
He chortled in his joy.*

*`Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.*

Now that it is brillig again, and everything is peaceful, or mimsy, or outgrabe, it is time to reflect on what we have covered. There is the ODS Escapechar to tell ods that a function is coming. Functions can be nested, and are scoped by using curly braces. Not all destinations are capable of all things. ODS Markup destinations which includes the new tagsets.rtf, html, Excelxp, LaTeX and many others, are unique in that the format functions are simply events within the tagsets, which means that any one can create new functions which can do almost anything. Think about if you have some special case where these capabilities might help streamline your processes.

CONTACT INFORMATION

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

Name: Eric Gebhart
Enterprise: SAS Institute
Address: SAS Campus Drive
City, State ZIP: Cary, NC, 27513
Work Phone: 919-632-1742
Fax:
E-mail: Eric.Gebhart@sas.com
Web: <http://support.sas.com/rnd/base/ods/odsmarkup>

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.