ODS Layout for RTF

A custom tagset

Richard A. DeVenezia, Independent Consultant, Remsen, NY

ABSTRACT

ODS LAYOUT is a preproduction feature in SAS® 9.0 through SAS 9.2. Preproduction means it is not officially supported nor fully developed. Regardless of its status, LAYOUT is a powerful tool for the PRINTER and PDF destination. It will let you “easily mix graphics, images, text, and tables, and arrange them on a page,” producing eye popping print or print copy. Hold it, wait, backup... PRINTER and PDF destination only? What about that RTF report I wanted to create?

This paper will demonstrate that the production ODS system has the ability to extend the standard RTF tagset to do what ODS LAYOUT does. Details of the Tagset and RTF languages will be discussed.

Keywords: ODS, LAYOUT, Template, TagSet, $options, RTF, pard, shp

INTRODUCTION

The ODS LAYOUT syntax for absolute positioning is as follows:

- Create a container

  ods layout start (absolute)
  [x=<dimension unit>]
  [y=<dimension unit>]
  [width=<dimension unit>]
  [height=<dimension unit>]
  ;

- Create a region within the container

  ods region
  [x=<dimension unit>]
  [y=<dimension unit>]
  [width=<dimension unit>]
  [height=<dimension unit>]
  ;

- Generate content within region

  proc print data=sashelp.class;
  or

  proc gplot data=sashelp.class;
  plot weight*age;

This paper will cover the implementation of tagsets.rtfex, an extension of destination tagsets.rtf, that handles absolute layout directives communicated through the ODS options() keyword.

## Absolute layout

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precise positioning</td>
<td>Overflow not visible</td>
</tr>
<tr>
<td>Numerous output areas</td>
<td>Areas can overlap</td>
</tr>
</tbody>
</table>

**rtfex** will mimic the functionality of ODS LAYOUT as far as regions on a page.

### TAGSET

A tagset is a program. A program that can respond to events and output content. Events that are generated (or fired) by ODS enabled SAS procedures (almost all of them) and ODS statements. Content that is formatted appropriately for the active destination. Output covers the gamut from Proc generated graphics, analytic results and reporting tabulations to embedded 'tags' utilized by third party applications that open the finished output.


A tagset is prepared under the aegis of Proc TEMPLATE.

```sas
Proc TEMPLATE;
 DEFINE TAGSET tagset-path / STORE=libref.template-store;
```

The tagset program code comes after the DEFINE statement. Code is organized into blocks; each block handles one event.

```sas
DEFINE EVENT event-name;
 ... tagset language statements ...
END;
```

### TAGSET VARIABLES

Four variable types are available in the tagset language:

- `$dictionary-variable['key']`
- `$list-variable<index>[
- `$scalar-variable`
- `$stream-variable`

These types are similar to DATA Step variable types in the following manner.

- *dictionary-variable* is a hash.
- *list-variable* is an array.
- *scalar-variable* is a variable.
- *stream-variable* is somewhat similar to a file.

The **SET** statement assigns a value to a variable or an indexed entry.

```sas
SET $name 'Richard'; * scalar;
SET $name[1] 'Richard'; * list;
SET $lastNameOf['Richard'] 'DeVenezia'; * dictionary;
```

### TAGSET STATEMENTS

Available tagset statements are as follows:

- `BLOCK` `BREAK` `CLOSE` `CONTINUE` `DELSTREAM` `DO` `DONE` `ELSE` `EVAL` `FLUSH` `ITERATE` `NDENT`
The highlighted statements are used in \texttt{rtfex}.

Statements can be conditionally executed by adding a slash (/) event-conditional after it. For example:

\begin{verbatim}
PUT 'Hello world' / if ($enthusiastic);
\end{verbatim}

\textbf{TAGSET EVENT-CONDITIONAL}

Available conditionals are as follows:

\textbf{ANY BREAK IF CMP CONTAINS EXIST IF NOT WHILE}

The highlighted statements are used in \texttt{rtfex}.

\textbf{ODDS AND ENDS}

The tagset language does take some getting used to. Some familiar coding forms are contrasted below.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{DATA Step} & \textbf{Tagset} \\
\hline
if condition then do; & do / if (condition); \\
...code... & ...code... \\
end; & done; \\
\hline
array X[10]; & iterate $X$; \\
do I = 1 to dim(X); & do / while _value_; \\
...code with X[I]... & ...code with _value_; \\
end; & next $X$; \\
& done; \\
\hline
\end{tabular}
\end{table}

The tagset language has much more depth than can be shown in this brief overview.

Developing a tagset can be frustrating at times because the language is different from the familiar DATA Step and the lack of an interactive debugger.

\textbf{ODS OPTIONS()}

The \texttt{OPTIONS()} option of the ODS statement allows a Base programmer to pass name-value pairs to a tagset. The tagset will receive an OPTIONS\_SET event and the name-value pairs will be in the automatic \$options[] dictionary-variable. For example the events fired by the BASE statement:

\begin{verbatim}
ODS tagsets.rtfex (name=Richard comment="This is my comment");
\end{verbatim}

can be handled with the following tagset code:

\begin{verbatim}
define event OPTIONS\_SET;
   putlog "name: " $options["name"];
   putlog "comment:" $options["comment"];
end;
\end{verbatim}

An ITERATE can be used to log all the options.

\begin{verbatim}
SET $guard 1;
ITERATE $options;
do /while _name_
   putlog _NAME_ = _VALUE_ ;
   putlog "$guard = "$ $guard;
NEXT $options;
eval $guard $guard+1;
stop /if $guard > 50;
done;
\end{verbatim}
TAGSET.RTFEX OPTIONS() SYNTAX

The custom tagset will mimic the functionality of ODS LAYOUT with respect to a page. There will be no outer container for regions; instead, regions are positioned directly on the page. This design choice was made to simplify the inner workings of the tagset code.

The name-value pairs that are processed by the tagset form a simple application programming interface (API) for the Base programmer. Namespace safety is ensured by passing all information through a single argument named REGION_ACTION.

<table>
<thead>
<tr>
<th>Verb</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN</td>
<td>Left, Top, Width, Height</td>
</tr>
<tr>
<td></td>
<td>Units are inches</td>
</tr>
<tr>
<td>CLOSE</td>
<td>none</td>
</tr>
</tbody>
</table>

Only one region can be open at a time. The following rules will be enforced by the tagset.

• OPEN can only be used if no region is open.
• CLOSE can only be used if a region is open.

The following tagset code polices the region action.

define event GRAND_INIT;
    set $REGION_CLOSED 'closed';
    set $REGION_OPEN   'open';
    set $REGION_STATE $REGION_CLOSED;
end;

define event OPTIONS_SET;
    set $action lowcase($options['REGION_ACTION']);
    break / if ^$action; * stop processing the event;
    unset $options['REGION_ACTION']; * remove action from options dictionary;
    set $verb scan($action,1,:);
    do / if $verb not in ('open', 'close');
        putlog "WARNING: Unknown verb " $verb;
        unset $verb;
        break;
    done;

CLOSE VERB

The region state is maintained in variable $REGION_STATE.

do / if cmp ($verb, 'close');
    do / if cmp($REGION_STATE, $REGION_OPEN);
        open body_tbl;
            put !!! rtf-codes to close an open region !!!
        close;
        set $REGION_STATE $REGION_CLOSED;
        putlog "NOTE: Region closed.";
    else;
        putlog "WARNING: Can not close a region because none is open.";
    done;
These three notable statements appear in the above code.

- **open** tells the tagset executor that PUTs should output to the body_tbl portion of ODS destination
- **put** outputs raw rtf codes
- **close** stops placing content in body_tbl and future PUTs get appended to output file

**OPEN VERB**

Ensure the rules about only one open region are followed.

```plaintext
* verb must be 'open' to reach here;
do / if cmp ($REGION_STATE, $REGION_OPEN);
   putlog "WARNING: Can not open a region because one is already open."
   unset $action;
   unset $verb;
   break;
done;
```

Extract the top, left, width and height settings and convert to rtf twip units (1,440 twips per inch). In the following code, note the use of `SCAN()`. A tagset program can use any function that is allowed in a SQL where clause. *Trivia: Internally, the tagset executor uses the SQL where executor!*

```plaintext
set $settings scan($action,2,':');
unset $action;

set $left  scan ($settings,1,',');
set $top    scan ($settings,2,',');
set $width  scan ($settings,3,',');
set $height scan ($settings,4,',');
```

The extracted settings are converted to numerics using a combination of `EVAL` and `INPUTN()`.

```plaintext
eval $x inputn($left  ,'best12. ');
eval $y inputn($top   ,'best12. ');
eval $w inputn($width ,'best12. ');
eval $h inputn($height,'best12. ');

do / if NMISS($x,$y,$w,$h);
   putlog "WARNING: " $settings
   " is not a comma separated list of left,top,width,height";
   unset $settings;
   break;
   done;
```

`EVAL` is used again to perform the unit conversion.

```plaintext
eval $left_twip  $x * 1440;
eval $top_twip    $y * 1440;
eval $right_twip $w * 1440 + $left_twip;
eval $bottom_twip $h * 1440 + $stop_twip;
unset $x; unset $y; unset $w; unset $h;
```
UNSET appears quite often; it should be used whenever a variable will not be used again in further code. The reason is that all tagset variables are global in scope and removing a variable with UNSET can prevent both unwanted side effects and hard to debug situations.

EVAL is used once more to perform integer rounding. The numerous EVAL steps are an artifact from development when PUTLOG was used to check intermediate results.

```sql
eval $left_twip int($left_twip);
eval $top_twip int($top_twip);
eval $right_twip int($right_twip);
eval $bottom_twip int($bottom_twip);
```

Once the twip values have been computed they can be used as part of the content output by the tagset:

```sql
open body_tbl;
  put "!!! rtf-codes containing twip values to open a new region !!!"
  close;

unset $xtwip; unset $ytwip; unset $wtwip; unset $htwip;

set $REGION_STATE $REGION_OPEN;
putlog "NOTE: Opened region " $settings;
unset $settings;
```

**RTF CODES**
The matter of which RTF codes to output at the open and close of a region remain a mystery. These codes are the secret sauce in the Big Mac® of the *rtfex* tagset and are surprisingly simple.

The core idea is to have the tagset generate RTF code that defines an absolute positioned text box and let the content (or text) of the box be generated by SAS Procedures.

- **OPEN** will emit RTF that defines box position and opens a `\shptxt`
- Normal SAS output for RTF destination
- **CLOSE** will emit closing RTF

Numerous sample documents containing text boxes were created using applications such as Microsoft Word®, Wordpad and OpenOffice.org Writer®. A careful study of the resultant rtf guided the author in which codes to use.

A detailed discussion of the rtf language elements selected for use is beyond the scope of this paper.

**OPEN \SHPTXT**
The unsettled matter of

```sql
put "!!! rtf-codes containing twip values to open a new region !!!"
```

becomes

```sql
put '({\*\comment_region_start}) nl;
put '\pard \widctlpar\adjustright' nl
'\shp' nl
'{{\shpinst' nl
'\shpbxpage' /* left and right relative to page */
'\shpbypage' /* top and bottom relative to page */
nl
'\shpleft' $left_twip
```

6
The highlighting denotes nested braces that need to be balanced when the region is closed. The use of the twip values is also highlighted.

CLOSE \SHPTXT
The unsettled matter of

```
put !!! rtf-codes to close an open region !!!
```

becomes

```
put '}}})' nl;
put '{*\comment_region_end}' nl;
```

SAMPLE PROGRAM
ods listing close;
goptions device=png;
ods tagsets.rtfex body="sample.rtf" style=journal;
title;footnote;
proc print data=sashelp.class(firstobs=1 obs=1);run;
ods tagsets.rtfex options(REGION_ACTION='open:5,1,3,2') startpage=no;
   proc print data=sashelp.class(firstobs=2 obs=2); run;
ods tagsets.rtfex options(REGION_ACTION='close');
ods tagsets.rtfex options(REGION_ACTION='open:1,4.5,3.5,3.5');
   proc print data=sashelp.class(firstobs=3 obs=3); run;
ods tagsets.rtfex options(REGION_ACTION='close');
ods tagsets.rtfex options(REGION_ACTION='open:4.5,3.5,3.5,3.5');
      goptions vsize=3in hsize=3in;
      proc gplot data=sashelp.class;
      plot weight * height; run; quit;
ods tagsets.rtfex options(REGION_ACTION='close');
ods tagsets.rtfex close;
ods listing;
MISCELLANEOUS
One feature of tagsets not discussed is the lack of an inheritance mechanism. In order to have rtfex function properly, the source of the original tagsets.rtf entry was duplicated and copied into rtfex source with a %include statement.

CONCLUSION
ODS LAYOUT is a powerful new feature coming to future releases of SAS. The tools found in Proc TEMPLATE and the tagset system may be used to emulate features of LAYOUT for output going to the RTF destination. The tagset language is a powerful subsystem that can be held in high regard.

CONTACT INFORMATION
Richard A. DeVenezia
9949 East Steuben Road
Remsen, NY 13438
(315) 831-8802
http://www.devenezia.com/contact.php

Richard is an independent consultant who has worked extensively with SAS products for over fifteen years. He has presented at previous SUGI, NESUG and SESUG conferences. Richard is interested in learning and applying new technologies. He is a SAS-L Hall of Famer and remains an active contributor to SAS-L.

The entire tagset source code will be available at the authors website, http://www.devenezia.com

RECOMMENDED READING


“ODS LAYOUT: Arranging ODS Output as You See Fit” - Schellenberger


ACKNOWLEDGMENTS
SAS is a trademark and registered trademark of SAS Institute in the USA and other countries.

Big Mac is a trademark and registered trademark of McDonald's in the USA and other countries.

Microsoft Word is a trademark and registered trademark of Microsoft Corporation in the USA and other countries.

OpenOffice.org is a trademark and registered trademark of Sun Microsystems, Inc., in the USA and other countries.