Paper 288-2011

Parlay Standard ODS HTML Pop-Up Boxes into Pop-Up Jackpots

Sean Hunt, Celtic Healthcare Inc., Mars, PA, USA

ABSTRACT

Effective dashboards balance the volume of displayed information with the inevitable need for additional (drill-down) data. One strategy to meet these goals is to employ pop-up boxes to convey drill-down data without having to navigate elsewhere. However, standard SAS® Output Delivery System (ODS) HTML output produces pop-up boxes with limitations (i.e., text formatting and display time). By injecting some simple JavaScript into SAS ODS output, the author will demonstrate examples of turning standard pop-up boxes into blank canvases for instantly displaying data tables and graphs. The same concepts can also be used to add dynamic menu systems and other functionality to standard ODS HTML output.

INTRODUCTION

The concept for this paper began with a question from my company's owner regarding our newly deployed dashboard. The question seemed simple enough: "Is there a way to keep the pop-up boxes from disappearing so quickly?" At that time, I was using a typical HTML pop-up box to store tons of additional data – in fact, much more than the users could comprehend before they disappeared. After researching a variety of options, I concluded that my only solution was to use some JavaScript that would allow a fully customizable pop-up box (or tool-tip). But, this approach also entailed a slight workaround to force standard SAS ODS HTML output to correctly contain the required JavaScript. Following a brief overview of the problem, this paper will use code samples to offer a dynamic solution for embedding JavaScript into SAS ODS HTML output. The target audience should have at least a basic working knowledge of SAS ODS graphical output and HTML code.

DASHBOARD OVERVIEW

The principles of effective dashboard design include minimizing clutter, using color sparingly, and condensing the information to fit on one screen. The latter can be the most difficult, particularly for an enterprise-level dashboard with dozens of indicators and layers. Dashboards should also engage the end-users, while inviting interactions for more in-depth analyses. This can be accomplished by including drill-down information and/or pop-up boxes that are activated by clicking directly on the dashboard charts, graphs, or data tables (Few 2006, p.171).

Thanks to the work of some SAS pioneers in the dashboard world, there are endless options for effectively conveying data. From creating individual graphical components such as bullet graphs and spark lines (SAS Institute, Inc.), to developing full-scale demos (Allison), there are numerous resources that include the SAS code necessary to get started. In particular, Robert Allison's dashboard gallery is a virtual treasure trove of working examples that are easy to comprehend and adapt.

My challenge was to use SAS/GRAPH to create a dashboard that could be updated daily and embedded as a Microsoft SharePoint Web Part. After several iterations of various key metrics and graphical formats, I unveiled an initial dashboard to our Executive Team. As alluded to above, I packed each and every pop-up box with a variety of additional data points. I achieved the goal of keeping users on the dashboard by having this information readily available without an additional mouse click. But, I inadvertently frustrated users with unattractive pop-up boxes that disappeared after a few seconds. After going back to the proverbial drawing board, I created a solution using some JavaScript. My second dashboard release contained customized pop-up boxes, complete with graphs and data tables – all of which stay visible until the user clicks elsewhere. As an added bonus feature, I included some additional JavaScript to enhance the navigation menus, including drop-down functionality.

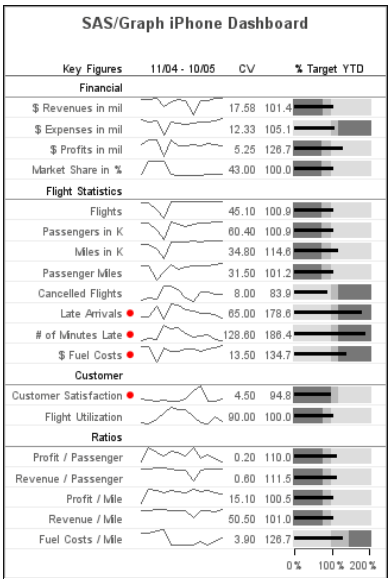

Figure 1. Dashboard Sample

ODS HTML OUTPUT

Let's now turn to some of the code behind the scenes. Standard SAS ODS HTML Output, with a little help from the SAS Annotate facility, allows you to create a Web-based version of basically any dashboard or graph that can be drawn by hand. A great example is Robert Allison's "iPhone Dashboard" (Figure 1). We'll start with this dashboard as a foundation, and then tweak the

code to include some customized pop-up boxes. Since this example was designed for viewing on a smartphone, it does not include any pop-up boxes or embedded links. Assuming we want to view this example in Internet Explorer, embedding some pop-up boxes would add value to an already exceptional design.

Pop-up boxes (with or without drill-down links) can be easily added by annotating an invisible box around a particular indicator, and then supplying the text we want the user to see when hovering over the box. The annotate code necessary to add a pop-up box to the first sparkline indicator ("Revenues") is below. Note that the code assumes we are already annotating a data set that contains one row per indicator to be output, including two variables named "measure" and "bar". Please refer to the references section for a link to the complete code.

```
data anno_table; set mydata;
…
xsys='1'; ysys='1';
function='move';
x=7;                                                    /* Left edge of sparkline */
y=(100/&lines)*y_line;                        /* Determines bottom edge of sparkline */
output;
function='bar'; line=1; size=1; style='empty';
x=x+(33);                                               /* Right edge of sparkline */
y=((100/&lines))*y_line + (100/&lines)        /* Determines top edge of sparkline */
length html $ 250;
html='title='||quote( trim(left(measure)) ||'0D'x||
trim(left(put(bar,percent8.1))) ||'0D'x ) ||' '||
'href="iPhone_dashboard.htm"';                                  /* Optional - link */
output;
…
run;
```

Now, we have a rather plain pop-up box that can display additional facts about an indicator. However, the only available formatting options are tabs and line breaks. This means that the most sophisticated output possible is a rudimentary data table. Furthermore, there is a limit to the number of characters that can be displayed (although the limit is quite generous). Plus, as mentioned earlier, the box will disappear after a few seconds. To display the box after it disappears, the user must move the mouse away from the target area and then back again, which makes it difficult to digest all of the information.

With the addition of some JavaScript, we can turn this pop-up box into something much more useful, not to mention visually appealing. But before getting into the JavaScript, let's compare the HTML code generated by SAS in the above examples to see what actually happened. This will be crucial later on for determining how to correctly call the necessary JavaScript functions.

In Figure 1, the entire dashboard is displayed as one image. The HTML code is fairly simple, consisting of a single "<img>" tag. Here is a portion of the actual HTML code, which just points to the PNG file that SAS created:

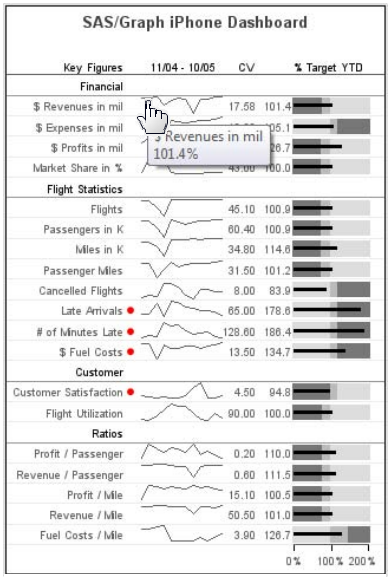

Figure 2. Standard Pop-Up Box

```
<img src="iphone_dashboard.png"
style=" background-color: #FFFFFF; height: 480px; width: 320px;"
border="0" class="c Graph">
```

In Figure 2, the dashboard is still displayed as a single image, but SAS added an image map to define the target areas for pop-up box. The "<img>" tag is basically the same, with the addition of the "usemap=" option:

```
<img src="iphone_dashboard2.png"
style=" background-color: #FFFFFF; height: 480px; width: 320px;"
border="0" usemap="#g53td4m4_map" class="c Graph">
```

Additionally, SAS added some code for the image map itself, consisting of a series of x-y coordinates and the text to be displayed in the tooltip:

```
<map name="g53td4m4_map">
<area shape="RECT" title="$ Revenues in mil 101.4%"
href="iPhone_dashboard.htm" coords="268,78,275,89">
```

Now that we've covered the basic anatomy of an HTML image map, we can move on to embedding some JavaScript.

JAVASCRIPT BASICS

To first step for adding JavaScript functions to your SAS output entails finding (and borrowing) some existing code. For the pop-up boxes, Walter Zorn's Web site contained exactly what I was looking for: an easy-to use blank canvas that would accept standard HTML formatting, such as bold fonts, tables, and even images (Zorn). After downloading the package and saving the "wz_tooltip.js" file in the same directory as the output files, you must insert the appropriate information into the body section of an HTML document. More specifically, insert the "<script>" tag right after the opening "<body>" tag. For the example to follow, this code does the trick:

```
<script type="text/javascript" src="wz_tooltip.js">
</script>
```

Now, we can look closer at the tooltip function itself. Essentially, each desired pop-up box needs three components:

1. The coordinates (or HTML link) that define the target area for the user to hover over
2. The text, images, and/or HTML table to display
3. Any desired formatting options

For a basic example, assume we want to add a custom pop-up to a standard HTML link, using all default options. First, within the HTML link tag, we just insert the behavior to perform with the users hovers on the link ("onmouseover") or exits the link ("onmouseout"). For this example, the desired behavior is a function called "TagToTip", which takes a single argument ("Span1"):

```
<a href="index.htm" onmouseover="TagToTip('Span1')" onmouseout="UnTip()">Link Text
</a>
```

Now, we also need to define the HTML element named "Span1":

```
<span id="Span1">This is some text for the pop-up box<br>
 with some <b>HTML formatting</b>
</span>
```

To add a custom pop-up to an image map, the same logic applies. We just need to add the same code within the "<area>" tag. Note that a number of options can be inserted here as well, such as width, height, and colors:

```
<area shape="RECT"
title="" href="insert_your_link_here.htm"
onMouseOver="TagToTip('Span2', OPTION1, 'OPTION1 VALUE', OPTION2, 'OPTION2 VALUE')"
onMouseOut="UnTip()"
coords="297,340,411,357">
```

Furthermore, we must define the "Span2" element, as above. Here is a sample HTML table definition, rather than just plain text:

```
<span id='Span2'>
 <table border='0' cellspacing='1' style='color:000044; font-size:8pt;'>
  <tr>
   <td>Table with some Text</td>
   <td>More Text</td>
  </tr>
 </table>
</span>
```

PUTTING IT ALL TOGETHER

The final step in the process is to get all of this code to function properly without any manual editing of the SAS output, with the ultimate goal being something like Figure 3 (below). But, the only place SAS output allows us to insert code is in the tooltip section, as we saw in Figure 1 (above). Let's go ahead and insert all of the needed code here to convert the standard pop-up box into a floating canvas containing an image file within an HTML table, and just see what happens. We're essentially just inserting the "html" as we did in Figure 2 (above), but here I've broken the components into what we need for the JavaScript function ("mouseover" and "span"), and then combined them together:

```
data anno_table; set mydata;
…
length html mouseover span $ 500;
mouseover='"'||"TagToTip('Span"||trim(left(data_order))||"',
 CLICKCLOSE, true,
 BGCOLOR, '#FFFFFF',
 SHADOW, true,
 BORDERCOLOR, '#707014',
 SHADOWCOLOR, '#999999')" || '"';
span="<span id='Span"||trim(left(data_order))||"'>"||
 "<table border='0' cellspacing='1' style='color:000044;font-size:8pt;'>"||
 "<tr><td colspan='2' align='left'><img src='popup" || trim(left(data_order)) ||
 ".gif'></td></tr>"||
 "<tr><td colspan='2' align='left'>Some interesting information.</td></tr>"||
 "</table></span><!--end tooltip content -->";
html= 'title="" href="#" onMouseOver='||trim(left(mouseover))||'
 onMouseOut="UnTip()"' || trim(left(span));
output;
html=""; span=""; mouseover="";
…
run;
```

If you were to render the code in your browser in this state, your dashboard would contain a few lines of ugly code, along with an error message. First of all, we're missing the "<script>" tag after the "<body>" tag. In addition, the HTML output needs to be rearranged somewhat to render properly. By reading and editing the HTML output file after SAS creates it, you can search for the tags you need to modify. By swapping a few pieces of code from within the HTML "<area>" tag to outside the tag, the pop-up boxes render beautifully when hovered upon.

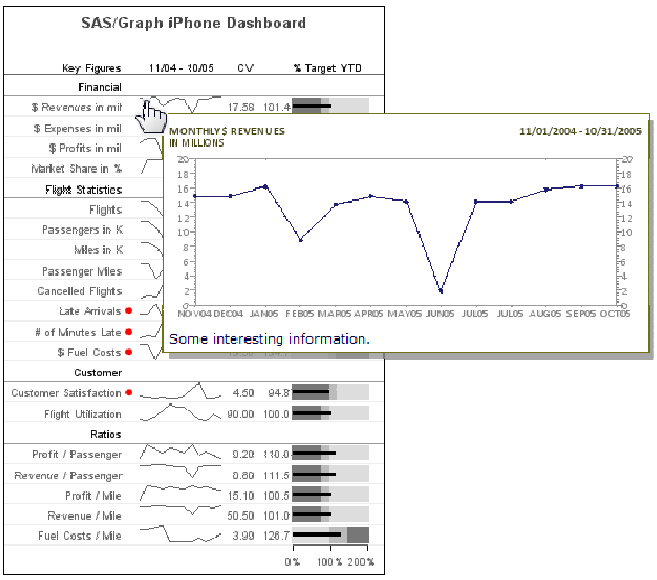

Figure 3. Customized Pop-Up Box

Let's take a closer look at what needs to happen:

<Script> Tag

We need the following HTML output:

```
<body onload="startup()"
onunload="shutdown()" class="Body">
```

to look like this:

```
<body onload="startup()" onunload="shutdown()" class="Body">
<script type="text/javascript" src="wz_tooltip.js"></script>
```

<Area> Tag

Furthermore, we need move the "" tag from the following HTML output to outside of the "<area>" tag:

```
<area shape="RECT" title="" href="#"
 onMouseOver="TagToTip('Span3', CLICKCLOSE, true,BGCOLOR, '#FFFFFF', SHADOW, true,
  BORDERCOLOR, '#707014', SHADOWCOLOR, '#999999')"
 onMouseOut="UnTip()"
<span id='Span3'>
<table border='0' cellspacing='1' style='color:000044;font-size:8pt;'>
 <tr><td colspan='2' align='left'><img src='popup3.gif'></td></tr>
 <tr><td colspan='2' align='left'>Some interesting Information.</td></tr>
</table></span><!--end tooltip content -->
coords="114,75,182,92">
```

Like this:

```
 <area shape="RECT" title="" href="#"
 onMouseOver="TagToTip('Span3', CLICKCLOSE, true,BGCOLOR, '#FFFFFF', SHADOW, true,
  BORDERCOLOR, '#707014', SHADOWCOLOR, '#999999')"
 onMouseOut="UnTip()"coords="114,75,182,92">
<span id='Span3'>
<table border='0' cellspacing='1' style='color:000044;font-size:8pt;'>
 <tr><td colspan='2' align='left'><img src='popup3.gif'></td></tr>
 <tr><td colspan='2' align='left'>Some interesting Information.</td></tr>
</table></span><!--end tooltip content -->
```

To accomplish our goal, we can import the HTML output file as a SAS data set, edit a few lines, and re-save as a new file. We can use a few SAS text functions to search for certain character strings to identify which lines need to be changed, and also extract any information that needs to be rearranged. The code below is definitely not optimal, but it reliably processes even large files with numerous pop-up boxes in less time than it takes SAS to create the output in the first place.

```
data temp;                          /* Save the data set for de-bugging purposes */
infile "c:\sas\sgf2011\iphone_dashboard3.htm" delimiter='0D'x DSD
 lrecl=32767 firstobs=1 sharebuffers;
file "c:\sas\sgf2011\iphone_dashboard3FINAL.htm" lrecl=32767;
informat text $2000.;
format temp1 temp2 temp3 $2000.;
retain switch 0;
input text $ @@;
if index(text,'<body')=1 then switch=1;
 else if switch=1 then do;
  if index(text,"<script")=1 then do;
   text='<script type="text/javascript" src="wz_tooltip.js"></script>'||
    trim(left(text));
   switch=0;
  end;
 end;
else if index(text,"onMouseOver")>0 then do;
 temp1=substr(text,1,index(text,"UnTip()")+7);
 temp2=substr(text,index(text,"UnTip()")+8,
  index(text,"coords")index(text,"UnTip()")-8);
 temp3=substr(text,index(text,"coords"),length(text)-index(text,"coords")+1);
 text=trim(left(temp1))||" "||trim(left(temp3))||" "||trim(left(temp2));
end;
if index(text,"</span")>0 and index(text,"</span>")=0 then
text=trim(left(text))||">";
put text;
run;
```

CONCLUSION

By leveraging custom HTML pop-up boxes, you can display a complete graph (including date and axis labels) without cluttering the main dashboard design. Thus, the additional information becomes available on-demand, and is less than one mouse click away.

While this paper only covers embedding JavaScript functions for creating customized pop-up boxes, I have also successfully used this method to add drop-down menus to SAS HTML output. Other applications are possible using the same approach. Generally speaking, the concept is find some JavaScript functions you'd like to use, determine what changes are necessary to SAS HTML output, get it to work by manually making the changes, then write some SAS code to do it for you.

Stated more simply, what happens in an HTML tag doesn't have to stay in the tag!

REFERENCES

Allison, R., "*Robert Allison's SAS/Graph Dashboards,*" at http://robslink.com/SAS/dashboards/aaaindex.htm (02/25/2011).

Allison, R., "*iPhone Dashboard,*" at http://www.robslink.com/SAS/democd38/iPhone_dashboard_info.htm (02/25/2011).

Few, Stephen. 2006. *Information Dashboard Design*. Sebastopol, CA: O'Reilly Media, Inc.

Zorn, W. "*JavaScript, DHTML Tooltips*," at http://gualtierozorni.altervista.org/tooltip/tooltip_e.htm (02/25/2011).

"SAS/GRAPH Dashboard Samples", SAS Institute Inc., at: http://support.sas.com/rnd/datavisualization/dashboards/ (02/25/2011).

CONTACT INFORMATION

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

Name: Sean Hunt
Enterprise: Celtic Healthcare, Inc.
Address: 150 Scharberry Ln
City, State: Mars, PA 16046
E-mail: hunts@celtichealthcare.com; seanhunt@emailplus.org