SCSUG 2018

DOMinate Your ODS Output with PROC TEMPLATE, ODS Cascading Style Sheets (CCS), and the ODS Document Object Model (DOM) by Mastering ODS TRACE DOM, Parsing CSS with a PROC TEMPLATE Front End, and Overriding ODS Style Inheritance with Customized Styles

Louise S. Hadden, Abt Associates Inc.

Troy Martin Hughes

ABSTRACT

SAS® practitioners are frequently required to produce SAS output in mandatory formats, such as using a company logo, corporate or regulated government templates, and/or cascading style sheet (CSS). SAS provides several tools to enable the production of customized output. Among these tools are the Output Delivery System (ODS) Document Object Model (DOM), CSS, PROC TEMPLATE, and ODS style overrides (usually applied in procedures and/or in originating data). This paper investigates "under the hood" of ODS destinations and PROC REPORT, and investigates how mastering ODS TRACE DOM and controlling styles using the CSSSTYLE option, PROC TEMPLATE, and style overrides can satisfy client requirements and enhance ODS output. Moreover, where report substance and style are maintained separately, data-driven software design is espoused because styles can be configured, tested, deployed, maintained, and modified without altering the underlying report content.

INTRODUCTION

SAS has provided myriad resources to control your output to meet standards and/or to look exactly as desired. The Output Delivery System (ODS) was one of the first steps in helping SAS practitioners to enhance the standard listing output. SAS has provided various output destinations to users throughout its history, including the listing destination, and in the not-so-recent past even specific printers. To facilitate user control over output styles, SAS has provided style and procedure templates that are applicable to all destinations, with portions of these templates that are specific to a given destination. SAS practitioners can further customize output by:

- Using cascading style sheets (CSS) as an option, or a supplement to, PROC TEMPLATE styles;
- Using PROC TEMPLATE to create a custom table template;
- Using PROC TEMPLATE to modify an existing style template;
- Using style overrides to enhance output with both procedural options (e.g. PROC REPORT define statements, etc.) and within incoming data streams (e.g. include style commands via ODS ESCAPECHAR);
- Using the ODS Document Object Model (DOM).

The purpose of this paper is to introduce SAS practitioners to some highly useful tools designed to facilitate an understanding of how SAS provides style information to control and enhance ODS output, and to incorporate CSS into SAS programs and ODS output. The ODS DOM became available as of SAS version 9.4, but the ability to incorporate CSS into ODS output as part of ODS code has been possible since SAS version 8.2 (and earlier unofficially!) A review of ODS vis a vis styles will be provided, and two primary methods of introducing CSS into ODS output will be discussed, along with the evolving functionality of CSS within SAS to accommodate different media types. Additionally, the role DOM can play in informing and controlling styles in ODS output will be explored. The paper is likely to appeal to intermediate SAS users and above.

FUNCTIONAL TOOLS

SAS provides many tools with which to create output, whether it is an output data set, spreadsheet, or a report destined for various types of media. The ODS and ODS Style Templates will be discussed briefly in the context of CSS and the ODS Document Object Model, along with macro processing and external reference files.

OUTPUT DELIVERY SYSTEM (ODS)

Most SAS users are intimately acquainted with the Output Delivery System (ODS) which debuted in SAS 7, and has evolved over time into a star-studded assemblage of useful (and varied) tools and destinations with which SAS users can produce stylish “documents” in a variety of formats and media. The basic “sandwich” technique of wrapping SAS procedure(s) in ODS destination open and end statements now has many options to enhance the production of, style of, and functionality of ODS output. These options include, but are not confined to: the ability to specify a SAS-provided or custom style template; the option to use ODS ESCAPECHAR to apply in-line styles within procedures and in incoming data; the ability to create a custom ODS table template; the ability to access CSS to influence output styles; the ODS TRACE capability to access information about ODS objects that are created; and last but not least, the subject of this paper and presentation, the ODS DOM. Further discussion of ODS as a whole is out of scope for this paper, although many useful and informative papers and books are available.

ODS Styles

To understand the impact of CSS and DOM on ODS output, the functionality of ODS styles will be explicated in more detail. One of the first ODS destinations was Hypertext Markup Language (HTML), and one of the cornerstones of HTML is the concept of “tags.” Historically, HTML (a text file) was designed to display data on the World Wide Web (W3) and combined content with style information using tags, or elements. Websites rendered HTML output, with style information built into each item to be displayed, into screen output. In the early days of ODS, SAS produced HTML 3.2 output designed to be viewed on a screen, although Microsoft Word® and Microsoft Excel® could open ODS HTML output as of Windows 97 so theoretically ODS HTML output could be printed. SAS provided a “default” style template to hold HTML output – if you did not specify a style template for HTML output, that default style, which had style instructions for given procedural output appropriate for screen output, would be used by SAS. The first ODS printer destinations quickly followed, producing ODS output in Rich Text File (RTF) and Portable Document Format (PDF) formats, which have their own proprietary methods of specifying style information in output files which are primarily designed for print media. Early on, SAS recognized that style information for different ODS destinations was conveyed in different ways, and accommodated those differences by developing different style templates that reflected the different media types. If a SAS user creates an RTF file and does not specify a style template, the default RTF style template is used. If a SAS user outputs a file to the HTML ODS destination and does not specify a style template, STYLES.HTMLBLUE is the default.

To further customize output, SAS practitioners can use PROC TEMPLATE to create a custom table template; use PROC TEMPLATE to modify an existing style template; use style overrides to enhance output with both procedural options (e.g. PROC REPORT define statements, etc.) and within incoming data streams (e.g. include style commands via ODS ESCAPECHAR); use CSS as an option, or a supplement to, PROC TEMPLATE styles; and use the ODS DOM. The focus of this paper is the use of CSS and DOM.

MACRO PROCESSING AND EXTERNAL REFERENCE TABLES

A complex report may have hundreds or thousands of cells, headers, and borders to style. It goes without saying that macro processing and external reference tables can play an essential role in customized SAS reporting. We’ll explore methods and examples below.

CONTENT, STRUCTURE, AND STYLE, OH MY!

Data products such as reports produced by SAS REPORT or TABULATE procedures are often customized to meet the style demands of analysts, customers, and other stakeholders. Static formatting might prescribe the layout, fonts, font colors, background colors, and other stylistic elements that are required for a specific organization, team, or report purpose. For example, a company might brand itself by requiring that reports (in addition to marketing and other published materials) contain their “trademark” colors — think McDonald’s yellow or A&W brown and orange. These stylistic decrees are often far-reaching within an organization and stable over time, thus they are more likely to be prescribed by human resources or marketing departments than by developers themselves.

Other stylistic elements, however, can be more dynamic and driven by the objective or content of a specific data product. For example, the font or font size of a report might be specified at runtime or dynamically

generated based on the quantity of data to be displayed, so long as it conformed to any organization-wide style guidelines that existed. In both cases, however, the flexibility and reliability of reports can be maximized where report content, structure, and style elements are separate and distinct. CSS, demonstrated in the following section, supports this flexibility, but it's beneficial to first identify these disparate elements—content, structure, and style.

Content

The *content* of a SAS data product is often a data set. For example, the following code produces the Students data set that includes an abbreviated list of Hogwarts School of Wizardry and Witchcraft students and their respective houses, as culled from the *Muggles' Guide to Harry Potter Characters*. The wizarding world of Harry Potter is particularly appropriate to demonstrate the transformative nature of style in SAS reporting, with differential house colors and atmosphere of magical transformation.

```
data students;
   infile datalines delimiter=',';
   length student $30 house $20;
   input student $ house $;
   datalines;
Neville Longbottom, Gryffindor
Draco Malfoy, Slytherin
Ginny Weasley, Gryffindor
Hannah Abbott, Hufflepuff
Tom Marvolo Riddle, Slytherin
Harry Potter, Gryffindor
;
```

A simple "report" having only content (thus, without modified structure or style) might be produced and is often sufficient for internal purposes. For example, the PRINT procedure having only the DATA parameter specified will by default print all variables within the Students data set:

```
proc print data=students;
run;
```

And this code produces the following output:

Obs	student	house
1	Neville Longbottom	Gryffindor
2	Harry Potter	Gryffindor
3	Ginny Weasley	Gryffindor
4	Hannah Abbott	Hufflepuff
5	Draco Malfoy	Slytherin
6	Tom Marvolo Riddle	Slytherin

Structure

But for many purposes, structure and style are additionally required for data products. *Structure* generally includes the order and formatting of report headers, rows, columns, and other elements, and will vary based

on the structure of the underlying data set. For example, within the abbreviated Students data set, Hogwarts houses are repeated but students are unique; however, are these business rules enforced through data quality constraints or is it possible for student names to be repeated as well? If the objective of a report is to produce an ordered list of unique students by house, data quality constraints and other business rules must be understood to ensure a report is correctly structured.

With this understanding (that student names are unique within the Students data set), the following REPORT procedure now conveys both *content* and *structure*, by grouping students by their respective houses, despite the original ordering in the underlying data set:

```
proc report data=students nocenter nowindows nocompletecols;
   column house student;
   define house / order 'Hogwarts House';
   define student / display 'Student';
run;
```

This REPORT procedure produces the following output having the default ODS style for the destination:

Hogwarts House	Student
Gryffindor	Neville Longbottom
	Harry Potter
	Ginny Weasley
Hufflepuff	Hannah Abbott
Slytherin	Draco Malfoy
	Tom Marvolo Riddle

Style

Content and structure are often sufficient to convey meaning, but *style* can really customize a stakeholder's experience. For example, just seeing this header with its ubiquitous light-blue background and nondescript, sans serif font causes my eyes to glaze over; it's apparent the report creator put no effort into customizing the style. Some personality can be bestowed simply by changing the font, font size, font color, and background color, as the following code demonstrates:

```
proc report data=students nocenter nowindows nocompletecols
      style(header)=[fontfamily="engravers mt" fontsize=4
      color=#FFFFFF backgroundcolor=#696969]
      style(column)=[fontfamily="century gothic" fontsize=3];
   column house student;
   define house / order 'Hogwarts House';
   define student / display 'Student';
run;
```

This modified report produces the following output using style overrides in the PROC REPORT commands:

HOGWARTS HOUSE	STUDENT
Gryffindor	Neville Longbottom
	Harry Potter
	Ginny Weasley
Hufflepuff	Hannah Abbott
Slytherin	Draco Malfoy
	Tom Marvolo Riddle

In other cases, stylistic elements can be used to convey meaning in addition to making things pretty. For example, fans of the Harry Potter series will immediately recognize that each of the four Hogwarts houses—Gryffindor, Slytherin, Hufflepuff, and Ravenclaw—has its own unique color branding. As illustrated in Figure 1, a Google search for "Hogwarts house color hex" yields the hexadecimal colors for each house.



Figure 1. Hexadecimal Colors for Hogwarts Houses

Each hex value represents a three-byte triplet that represents the red-green-blue (RGB) values for the specific hue. For example, the Gryffindor red is represented by 7F0909, or 7F red (127 in decimal), 09 green (9 in decimal), and 09 blue (9 in decimal). And, in addition to (or in lieu of) *reading* a house's name, report consumers might also benefit from *visualizing* house names in a report that shows house colors.

House colors, as specified in Figure 1, could be specified in STYLE statements in the COMPUTE block of PROC REPORT that are invoked with IF-THEN-ELSE conditional logic:

```
proc report data=students nocenter nowindows nocompletecols
```

```
      style(header)=[fontfamily="engravers mt" fontsize=4
     color=#FFFFFF backgroundcolor=#696969]
     style(column)=[fontfamily="century gothic" fontsize=3];
     column house student dummy;
   define house / order 'Hogwarts House';
   define student / display 'Student';
   define dummy / computed noprint;
   compute dummy;
      if house='Gryffindor' then call define('_c1_','style',
         'style=[backgroundcolor=#7F0909 color=#FFC500]'); else
      if house='Slytherin' then call define('_c1_','style',
         'style=[backgroundcolor=#0D6217  color=#AAAAAA]');  else
      if house='Hufflepuff' then call define('_c1_','style',
         'style=[backgroundcolor=#EEE117 color=#000000]'); else
      if house='Ravenclaw' then call define('_c1_','style',
         'style=[backgroundcolor=#000A90   color=#946B2D]');
      endcomp;
run;
```

This report now includes static style elements that guide the entire report as well as dynamic style elements that are driven by the data content:

HOGWARTS HOUSE	STUDENT
Gryffindor	Neville Longbottom
	Harry Potter
	Ginny Weasley
Hufflepuff	Hannah Abbott
Slytherin	Draco Malfoy
	Tom Marvolo Riddle

One weakness, however, is that these stylistic elements are commingled with the report structure. In other words, the house colors should be stable over time so it's inefficient (and error-prone) to require that their hex codes (and conditional logic) be repeated across various processes and programs. One method to overcome this weakness is to define the color schemes within SAS formats and to apply those formats within the REPORT procedure:

```
proc format;
   value $ houseback
   'Gryffindor'='#7F0909'
   'Slytherin'='#0D6217'
   'Hufflepuff'='#EEE117'
   'Ravenclaw'='000A90';
run;

proc format;
```

```
   value $ housefore
   'Gryffindor'='#FFC500'
   'Slytherin'='#AAAAAA'
   'Hufflepuff'='#000000'
   'Ravenclaw'='946B2D';
run;

proc report data=students nocenter nowindows nocompletecols
      style(header)=[fontfamily="engravers mt" fontsize=4 color=white
      backgroundcolor=peru]
      style(column)=[fontfamily="century gothic" fontsize=3];
   column house student dummy;
   define house / order 'Hogwarts House';
   define student / display 'Student';
   define dummy / computed noprint;
   compute dummy;
      call define('_c1_','style',
           'style=[backgroundcolor=$houseback. color=$housefore.]');
      endcomp;
run;
```

The output produced is identical to the previous REPORT procedure. As one final stylistic change, the house colors can be applied to the House column even for cells that don't contain the House variable. For example, to display Gryffindor colors on the first three rows of data (rather than just the first), the STYLE statement can be moved from the COMPUTE block to the DEFINE statement for the House variable:

```
proc report data=students nocenter nowindows nocompletecols
      style(header)=[fontfamily="engravers mt" fontsize=4
      color=#FFFFFF backgroundcolor=#696969]
      style(column)=[fontfamily="century gothic" fontsize=3];
   column house student;
   define house / order 'Hogwarts House'
style(column)=[backgroundcolor=$houseback. color=$housefore.];
   define student / display 'Student';
run;
```

This report is stylistically similar but now shows the house colors on every row, again relying on the Houseback and Housefore formats:

HOGWARTS HOUSE	STUDENT
Gryffindor	Neville Longbottom
	Harry Potter
	Ginny Weasley
Hufflepuff	Hannah Abbott
Slytherin	Draco Malfoy
	Tom Marvolo Riddle

One final improvement could extract the remaining dynamic style elements (e.g., font, font size, and header) from the report structure, thus more fully separating content, structure, and style elements. By placing the REPORT procedure within a macro and by parameterizing report style elements, these parameters can be specified at macro invocation. For example, the following macro now includes six parameters: HeaderFont, HeaderFontSize, HeaderFontColor, HeaderFontBackColor, ColFont, and ColFontSize:

```
%macro report(HeaderFont= /* Header Font Family */,
   HeaderFontSize= /* Header FontSize */,
   HeaderFontColor= /* Header Color */,
   HeaderFontBackColor= /* Header BackgroundColor */,
   ColFont= /* Column FontFamily */,
   ColFontSize= /* Column FontSize */);
proc report data=students nocenter nowindows nocompletecols
      style(header)=[fontfamily="&HeaderFont"
      fontsize=&HeaderFontSize color=&HeaderFontColor
      backgroundcolor=&HeaderFontBackColor]
      style(column)=[fontfamily="&ColFont" fontsize=&ColFontSize];
   column house student;
   define house / order 'Hogwarts House'
      style(column)=[backgroundcolor=$houseback. color=$housefore.];
   define student / display 'Student';
run;
%mend;

%report(HeaderFont=engravers
   mt, HeaderFontSize=4,
   HeaderFontColor=#FFFFFF,
   HeaderFontBackColor=#696969,
   ColFont=century gothic,
   ColFontSize=3);
```

The REPORT macro and this invocation produce an identical report. However, the style elements have now been removed from the report structure, so they can be specified when the macro is invoked. This data-driven solution maximizes flexibility because report attributes can be changed without the need to modify the underlying code. However, because the control data—represented both in the FORMAT procedures and the parameters—occur within the same program as the macro, the modularity of this solution can still be improved. To support greater software modularity, data independence, and interoperability, this solution is expanded and strengthened in the following section.

STYLISTIC TOOLS

Cascading Style Sheets (CSS)

Cascading style sheets (CSS) are external files (similar to a format catalog) that contain label-value pairs that can be used to describe various aspects of output, such as fonts, colors (foreground, background), and borders. Originally, CSS emerged as a way to apply style information to output presented on the W3, i.e., HTML. Prior to the implementation of CSS, the types of style information (and more) described previously were embedded along with the content being presented.

CSS files define stylistic elements within documents such as HTML web pages. Like the previous data-driven solution, the implementation of CSS enables style attributes to be separated from report content and structure. However, CSS is a preferred solution because CSS files are raw text, can be read by countless software applications (including SAS), and thus are more interoperable. For example, the Hogwarts colors are defined within two SAS FORMAT procedures, enabling other instances of the REPORT procedure or even other SAS procedures to rely on these control data. And, by saving the user-defined Houseback and Housefore formats to a permanent format library, the formats can be referenced in the future without the need to rerun the FORMAT procedures.

However, in the preceding solution, house colors are maintained only in SAS formats which are of no use to other software applications. For example, Hogwarts School of Wizardry and Witchcraft likely would want to use its branded house colors not only in SAS reports but also in PDFs, web pages, and other published documents that are unrelated to SAS. This interoperability is possible because a single CSS file can drive the dynamic styling of electronic media across an entire organization. Moreover, the maintenance of a single trusted repository for these style control data support master data management (MDM) best practices. Thus, the integrity of a single CSS file is greater because multiple (possibly asynchronous) copies of style data are not being maintained. MDM also supports greater maintainability, because when a style must be changed, it can be changed in a single location to alter stylistic elements in all derivative data products and documents that rely on that CSS file.

The full extent of CSS functionality far exceeds the bounds of this text, but this brief introduction offers a CSS-based solution that is functionally equivalent to the previous SAS-only solution[1]. Thus, both static style attributes (such as the Hogwarts house colors) as well as more dynamic style attributes (such as the report font, font size, and font color) can be defined within CSS files. In fact, behind the scenes, the REPORT procedure is already using CSS to represent stylistic elements—so the only task is to understand how to directly, rather than indirectly, rope and wrangle these controls.

To peel back the covers on the REPORT procedure, the preceding Hogwarts roster can be saved to an HTML file with the following code:

```
%let loc=/folders/myfolders;
ods html path="&loc" file="hogwarts.html";
proc report data=students nocenter nowindows nocompletecols
     style(header)=[fontfamily="engravers mt" fontsize=4
     color=#FFFFFF backgroundcolor=#696969]
     style(column)=[fontfamily="century gothic" fontsize=3];
     column house student;
   define house / order 'Hogwarts House'
      style(column)=[backgroundcolor=$houseback. color=$housefore.];
   define student / display 'Student';
run;
ods html close;
```

Then REPORT is run and the HTML file is opened in WordPad, the critical role of CSS is apparent in just the first few lines:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta name="Generator" content="SAS Software Version 9.4, see www.sas.com">
<meta http-equiv="Content-type" content="text/html; charset=utf-8">
<title>SAS Output</title>
<style type="text/css">
<!--
.activelink
```

---

1 The w3schools offers a fantastic introduction to CSS at https://www.w3schools.com/css/css intro.asp and W3C maintains the current CSS standards at https://www.w3.org/standards/techs/css#w3c all.

```
{
  color: #800080;
}
```

Thus, the text that follows the “text/css” section defines the styles to be used in the later report. This method of infusing CSS into documents is referred to as embedded CSS, whereas CSS can also be referenced in which the CSS elements are maintained in an external CSS file. To understand how the REPORT procedure relies on CSS, the style elements that were specified within REPORT (e.g. Engravers MT font, font size 4, white lettering color, etc.) can be searched within the Hogwarts.html file.

SAS documentation also introduces CSS, including how to implement CSS within the output display system (ODS). One of the most useful SAS documentation rubrics, reproduced here as Table 1, demonstrates how CSS selectors map to HTML tags and ultimately to SAS data products.

Class selector	Tag	What It Is Responsible For
.ContentTitle	<SPAN>	The title in the Table of Contents
.NoteContent	<TD>	The line statement with PROC REPORT
.Output	<TABLE>	Attributes on the table tag
.Graph	<IMG>	The images or graphs
.ProcTitle	<TD>	The procedure name on the body file
.Data	<TD>	The cell values
TD	<TD>	All numeric or right-justified data
.Table	<TABLE>	The table
.ContentFolder	<UL>	Overall items in the table of contents
.Byline	<TD>	The BY values
.PagesProcLabel	<LI>	The procedure name in the table of pages
.Header	<TD>	The column headers
.ContentProcLabel	<LI>	The procedure name when the ODS PROCLABEL statement has been used
.PagesTitle	<SPAN>	The title on the table of pages
.PagesProcName	<LI>	The procedure name on the table of pages
.ContentItem	<A><DT><LI>	The leaf or item in the table of contents; this is the hyperlink that you click
.Body	<BODY>	The body file, which renders such things as the background color and the margins
.DataEmphasis	<TD>	The summary line with PROC REPORT
.ContentProcName	<LI>	The procedure name in the table of contents
.PagesItem	<A><DT><LI>	The leaf or node in the table of contents; this is the link that you click
.RowHeader	<TD>	The row headers
.SystemTitle	<TD>	The SAS System titles
.SystemFooter	<TD>	The SAS footnotes

Table 1. CSS Selectors and HTML Tags

External CSS files can be referenced with the CSSSTYLE parameter within the ODS statement. For example, the following ODS statement and subsequent REPORT procedure now reference style elements contained within Hogwarts_style.CSS:

```
%let loc=/folders/myfolders;
ods html path="&loc" file="hogwarts.html" cssstyle="&loc/hogwarts_style.css";
proc report data=students nocenter nowindows nocompletecols;
   column house student;
   define house / order 'Hogwarts House'
      style(column)=[backgroundcolor=$houseback. color=$housefore.];
   define student / display 'Student';
run;
ods html close;
```

This code runs when the following CSS file is saved to Hogwarts_style.css:

```
.Header
```

```
{
  font-family: engravers mt;
  font-size: 4;
  color: #FFFFFF;
  background-color: #696969;
}
.Data
{
  font-family: century gothic;
  font-size: 3;
  color: #000000;
  background-color: #FFFFFF;
}
```

This code and referenced CSS file produce the following output, which is slightly different. This occurs because some of the style attributes (such as borders, cell padding, and cell widths) were inferred from defaults where explicit arguments were not specified:

HOGWARTS HOUSE	STUDENT
Gryffindor	Neville Longbottom
	Harry Potter
	Ginny Weasley
Hufflepuff	Hannah Abbott
Slytherin	Draco Malfoy
	Tom Marvolo Riddle

By modifying the borders and cell padding, the table is looking fairly identical to the original produced in the SAS-only solution. Also note that the BORDER-COLLAPSE property does need to be set to COLLAPSE so that adjoining cells share a single border line. The revised CSS file follows:

```
.Header
{
  font-family: engravers mt;
  font-size: 4;
  color: #FFFFFF;
  background-color: #696969;
  border: 1px #C0C0C0 solid;
  padding: 10px;
}
.Data
{
  font-family: century gothic;
  font-size: 3;
  color: #000000;
  background-color: #FFFFFF;
  border: 1px #C0C0C0 solid;
}
.Table
{
  border-collapse: collapse;
```

```
}
td, td
{
  padding: 10px;
}
```

Without any modifications to the code, the REPORT procedure can be rerun and produces the following output:

HOGWARTS HOUSE	STUDENT
Gryffindor	Neville Longbottom
	Harry Potter
	Ginny Weasley
Hufflepuff	Hannah Abbott
Slytherin	Draco Malfoy
	Tom Marvolo Riddle

Note that the revised CSS file can be reused to drive style elements in other procedures and data products. In addition, the dynamic elements (background and foreground color of the Hogwarts House column) can be assigned using <DIV, </DIV> and conditional data-status commands in your CSS file.

SAS, CSS AND ODS TRACE DOM

CSS AND SAS

Prior to SAS 9.2, PROC TEMPLATE was the only available tool in SAS to customize styles. As of SAS 9.2, SAS has allowed the use of both internal and external CSS files with three types of ODS output, HTML, PDF and RTF. At that point, SAS was able to read, or parse, CSS files and convert them into PROC TEMPLATE syntax. In SAS 9.3, SAS more fully integrated the CSS implementation in ODS so that context-based CSS selectors are possible in all types of ODS output that use styles. SAS is currently the only software which allows the integration of CSS in PDF, native EXCEL and RTF output.

SAS practitioners can call an external CSS file into play using a CSSSTYLE= option on an ODS destination call, similar to a STYLE= option for a SAS style template. We've demonstrated an example of the most recent method above:

```
ODS DESTINATION CSSSTYLE="FILEREF/URL/FTP protocol/HTTP protocol";

      Procedures...

ODS DESTINATION CLOSE;
```

CSS files perform in ODS just as a SAS- or user- produced ODS style template performs, allowing users to edit existing external CSS files or create new ones to develop a valid SAS style template on demand. Benefits of this facility are obvious, including standard, documented CSS syntax and tools and the additional control and flexibility over ODS output achieved via an external configuration file.

CSS developers have added @ MEDIA rules to the CSS2 specification to accommodate the burgeoning list of media types, including aural, Braille, TV, TTY, projection, etc. as well as print and screen. SAS

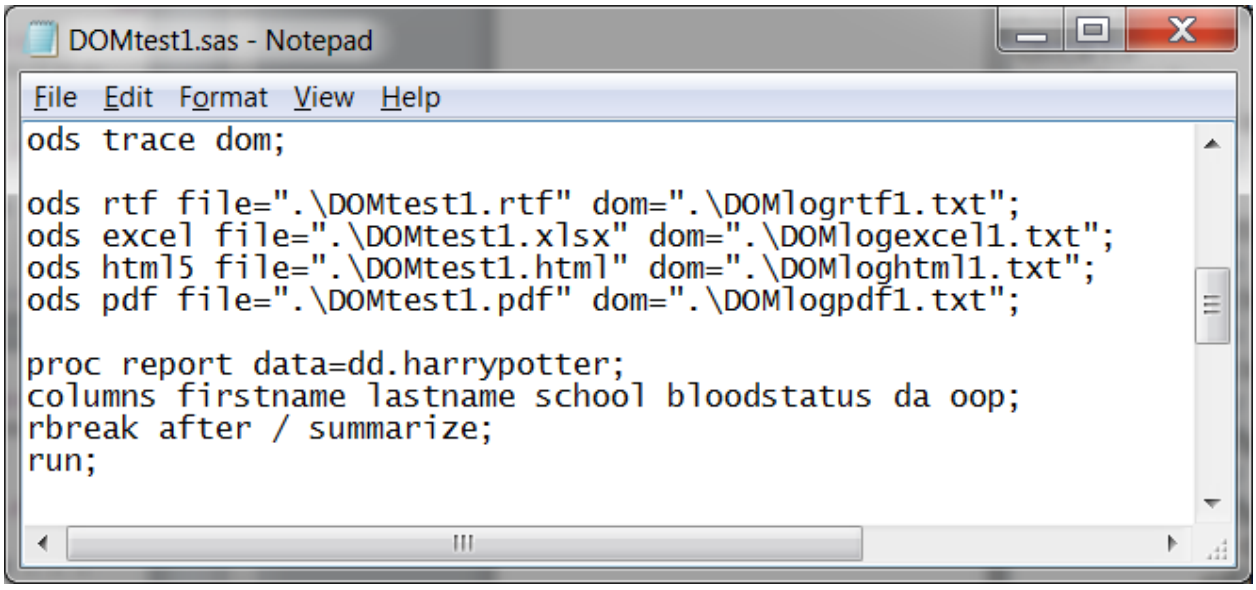
developers have followed suit, focusing on the print and screen media types that apply most readily to SAS destinations.
SAS ODS is not able to use all of the style elements that potentially occur in a CSS file – only a subset of style possibilities catalogued in the CSS 2.1 specification is available. To fully realize the potential of CSS in SAS ODS, users must inform themselves of what CSS style elements will work with which destination, procedure, and media types without manually comparing style sheets and templates line by line.

As noted, there are many non-SAS users and uses of CSS. We've seen several examples of styles that CSS can modify – but what if you want to find out what ODS output "objects" can be modified with a CSS when you can't view the output in a line editor (i.e. PDF) or don't want to wade through millions of lines of output? SAS provides tools such as DOM (the subject of this paper) and the TAGSETS.STYLE_POPUP or TAGSETS.ODSSTYLE destinations that allow users to identify the necessary style items for output.

ODS TRACE DOM

ODS TRACE is a very useful statement that allows SAS practitioners to trace the output created by a given procedure (and different procedural options), e.g. PROC CONTENTS. Additionally, SAS has provided a way to "trace" how styles are applied in different destinations, and what styles are able to be modified. ODS TRACE DOM produces an in-memory representation of an ODS report. As with other ODS TRACE, ODS TRACE DOM provides information in the log – in this case about ODS style elements. You can use the DOM to access and update the content, structure, and style of ODS output dynamically. By viewing the DOM, you can determine what elements and attributes exist so that you can construct CSS selectors to access those areas. To view the DOM for a destination, use the DOM option in any ODS destination statement except ODS LISTING and ODS OUTPUT. You can also specify that a listing of the DOM go to a specified text file, by specifying DOM= on the ODS destination call – while ODS TRACE DOM goes to the log.

The syntax follows:



A screenshot of the resulting text file is shown below, followed by the log representation.

DOMlogexcel1a.txt - Notepad

File Edit Format View Help

```
<!DOCTYPE html>
<html>
   <head>
      <title>ODS EXCEL DOM</title>
      <link rel="stylesheet" href="style.css">
   </head>
   <body dest="excel" class="body">
      <div>
      </div>
      <div>
      </div>
      <body class="startupfunction">
      </body>
      <body class="shutdownfunction">
      </body>
      <div class="body">
         <div>
         </div>
         <section id="idx" class="oo" data-name="procreporttabl
```

DOMtest1.log - Notepad

File Edit Format View Help

```
16          ods trace dom;
17
18          ods rtf file=".\DOMtest1.rtf" dom=".\DOMlogrtf1.txt";
NOTE: Writing RTF Body file: .\DOMtest1.rtf
<!DOCTYPE html>
<html>
   <head>
      <title>ODS RTF DOM</title>
      <link rel="stylesheet" href="style.css">
   </head>
   <body dest="rtf" class="body">
      <presentation class="tableheadercontainer">
         <presentation class="tablefootercontainer">
            <presentation class="columngroup">
19          ods excel file=".\DOMtest1.xlsx" dom=".\DOMlogexcel1.txt";
<!DOCTYPE html>
<html>
   <head>
      <title>ODS EXCEL DOM</title>
      <link rel="stylesheet" href="style.css">
   </head>
   <body dest="excel" class="body">
      <div>
      </div>
      <div>
      </div>
      <body class="startupfunction">
      </body>
      <body class="shutdownfunction">
      </body>
```

Note that two different ODS destinations are used for the examples above. The DOM varies with different destinations – not all style commands are available in all destinations.

CONCLUSION

CSS, style templates (SAS provided and custom), and style overrides can enhance SAS Output Delivery System (ODS) output. The ODS Document Object Model (DOM) acts as a liaison between ODS and cascading style sheets (CSS). DOM translates procedural output created with CSS into both PROC TEMPLATE language and properly rendered data in most SAS destinations that can be edited. Thus, looking at the output of DOM and CSS can inform and facilitate the development of custom styles in ODS. SAS has always been at the forefront of data reporting and visualization, from the early days of ODS reporting. Create ODS exactly where you want it, how you want it, and when you want it, using CSS and the ODS DOM.

REFERENCES

Base SAS: SAS Notes and Concepts for ODS. Cary, North Carolina: SAS Institute. https://support.sas.com/rnd/base/ods/templateFAQ/Template_csstyle.html.

Bauerly, Kerril S. 2000. "Style Sheets, Javascript, and SAS??? Oh, My!". Proceedings of the SAS Users Group International, Indianapolis, IN. http://www2.sas.com/proceedings/sugi25/25/iw/25p185.pdf

Benjamin Jr., William E. 2017. "Working with the SAS® ODS EXCEL Destination to Send Graphs, and Use Cascading Style Sheets When Writing to EXCEL Workbooks". Proceedings of the Western Users of SAS Software, Long Beach, CA. http://www.lexjansen.com/wuss/2017/137_Final_Paper_PDF.pdf

Davidson, Kevin and Duong, Minh. 2010. "Using Dynamic and Cascading Prompts in SAS® Enterprise Guide®". Proceedings of the SAS Global Forum, Seattle, WA. http://support.sas.com/resources/papers/proceedings10/041-2010.pdf

Durie, Ted. 2005. "ODS HTML and CSS". Proceedings of the Western Users of SAS Software, San Jose, CA. http://www.lexjansen.com/wuss/2005/data_presentation/dp-sas_ods_html_and_css.pdf

Eslinger, Jane. 2016. The SAS Programmer's PROC REPORT Handbook: Basic to Advanced Reporting Techniques. Cary, NC: SAS Institute Inc.

Eslinger, Jane. 2018. The SAS Programmer's PROC REPORT Handbook: ODS Companion. Cary, NC: SAS Institute Inc.

Eubanks, Elizabeth and Trahan, Shane. 2013. "An Integrated Approach to Codebook Generation Using SAS®, HTML/CSS, and the .NET Framework". Proceedings of the SAS Global Forum, San Francisco, CA. http://support.sas.com/resources/papers/proceedings13/259-2013.pdf

Fehd, Ronald. 2003. "A Journeyman's reference: The Writing for Reading SAS Style Sheet: Tricks, Traps, Tips, and Templates, from SAS-L Macro Maven". Proceedings of the South East SAS Users Group, St. Pete Beach, FL. http://analytics.ncsu.edu/sesug/2003/AD08-Fehd.pdf

Fehd, Ronald. 2003. "The Writing for Reading SAS® Style Sheet: Tricks, Traps, and Tips From SAS-L's Macro Maven". Proceedings of the South Central SAS Users Group, Houston, TX. http://www.lexjansen.com/scsug/2003/Fehd -- ARRAY.pdf

Hatcher, Diane and Hsueh, LanChien. 2009. "Dynamic Prompts Make Data Cascading Easy: Introducing New Features in SAS® 9.2 Prompt Framework". Proceedings of the SAS Global Forum, National Harbor, MD. http://support.sas.com/resources/papers/proceedings09/355-2009.pdf

Khurshed, Fareeza. 2011. "Creating a Table of Contents for Microsoft Word Using AutoFormat and Cascading Style Sheets". Proceedings of the SAS Global Forum, Las Vegas, NV. http://support.sas.com/resources/papers/proceedings11/116-2011.pdf

Liebhardt, Ingo. 2010. "Personalized Web Reports in Style: Tweaking SAS® ODS, Tagsets, and CSS to Get the Output Right". Proceedings of the SAS Global Forum, Seattle, WA. http://support.sas.com/resources/papers/proceedings10/221-2010.pdf

Mann, Robert and and Zalcman, Rosely Flam. 2013. "Creating a Management-Friendly HTML Report Using SAS® ODS Markup, Style Sheets, and JavaScript". Proceedings of the SAS Global Forum, San Francisco, CA. http://support.sas.com/resources/papers/proceedings13/238-2013.pdf

McCoy, Sheryl and Nilsson, Mary. 2014. "PhUSE Computational Science Symposium (CSS) White Paper on Analyses and Displays Associated with Adverse Events". Proceedings of the FDA/PhUSE US CSS, Silver Spring, MD. http://www.phusewiki.org/docs/2014/CSS/PP17 Final Poster.pdf

Muggles' Guide to Harry Potter Characters. https://en.wikibooks.org/wiki/Muggles%27_Guide_to_Harry_Potter/Characters

Pandya, Niraj J.. 2011. "Get Dynamic Multi-sheet Excel Workbook with STYLE using ODS". Proceedings of the PharmaSUG, Nashville, TN. http://www.lexjansen.com/pharmasug/2011/PO/PharmaSUG-2011-PO06.pdf

Parker, Chevell. 2003. "Generating Custom Excel Spreadsheets using ODS". Proceedings of the PharmaSUG, Miami, FL. http://www.lexjansen.com/pharmasug/2003/SASInstitute/sas129.pdf

Parker, Chevell. 2003. "Generating Custom MS EXCEL Spreadsheets Using ODS". Proceedings of the Northeast SAS Users Group, Washington, DC. http://www.lexjansen.com/nesug/nesug03/gv/s1054.pdf

Parker, Chevell. 2006. "Now - That's Your Style!!!!!". Proceedings of the South East SAS Users Group, Atlanta, GA. http://analytics.ncsu.edu/sesug/2006/DP10_06.PDF

Parker, Chevell. 2009. "Creating That Perfect Data Grid Using the SAS® Output Delivery System". Proceedings of the SAS Global Forum, National Harbor, MD. http://www2.sas.com/proceedings/forum2008/258-2008.pdf

Parker, Chevell. 2009. "The ODS Menu for All Appetites and Applications". Proceedings of the PharmaSUG, Portland, OR. http://www.lexjansen.com/pharmasug/2009/sa/SA-AD-01.pdf

Parker, Chevell. 2010. "A SAS® Output Delivery System Menu for All Appetites and Applications". Proceedings of the Western Users of SAS Software, San Diego, CA. http://www.lexjansen.com/wuss/2010/Applications/3028_2_APP-Parker1.pdf

Parker, Chevell. 2010. "The Ten Most Frequently Asked Questions about SAS® to Excel". Proceedings of the Midwest SAS Users Group, Milwaukee, WI. http://www.lexjansen.com/mwsug/2010/excel_db/MWSUG-2010-113.pdf

Parker, Chevell. 2010. "Using SAS® Output Delivery System (ODS) Markup to Generate Custom Pivot Tables and Pivot Charts". Proceedings of the Northeast SAS Users Group, Baltimore, MD. http://www.lexjansen.com/nesug/nesug10/ad/ad09.pdf

Parker, Chevell. 2010. "Using SAS® Output Delivery System (ODS) Markup to Generate Custom PivotTable and PivotChart Reports". Proceedings of the Western Users of SAS Software, San Diego, CA. http://www.lexjansen.com/wuss/2010/Applications/3029_2_APP-Parker2.pdf

Parker, Chevell. 2011. "Let's Give 'Em Something to TOC about: Transforming the Table of Contents of Your PDF File". Proceedings of the Western Users of SAS Software, San Francisco, CA. http://www.lexjansen.com/wuss/2011/datapresentation/Papers_Parker_C_76204.pdf

Parker, Chevell. 2011. "The Perfect Marriage: The SAS® Output Delivery System (ODS) and Microsoft Office". Proceedings of the Western Users of SAS Software, San Francisco, CA. http://www.lexjansen.com/wuss/2011/datapresentation/Papers_Parker_C_76200.pdf

Parker, Chevell. 2013. "The SAS® Output Delivery System: Boldly Take Your Web Pages Where They Have Never Gone Before!". Proceedings of the SAS Global Forum, San Francisco, CA. http://support.sas.com/resources/papers/proceedings13/017-2013.pdf

Parker, Chevell. 2015. "Secrets from a SAS Technical Support Guy: Combining the Power of the SAS® Output Delivery System with Microsoft Excel Worksheets". Proceedings of the Western Users of SAS Software, San Diego, CA. http://www.lexjansen.com/wuss/2015/156_Final_Paper_PDF.pdf
Parker, Chevell. 2015. "Staying Relevant in a Competitive World: Using the SAS® Output Delivery System to Enhance, Customize, and Render Reports". Proceedings of the Midwest SAS Users Group, Omaha, NE. http://www.lexjansen.com/mwsug/2015/BI/MWSUG-2015-BI-04.pdf

Parker, Chevell. 2017. "A Ringside Seat: The ODS Excel Destination versus the ODS ExcelXP Tagset". Proceedings of the South Central SAS Users Group, Dallas, TX. http://www.lexjansen.com/scsug/2017/Chevell_SCSUG.pdf

Parker, Chevell. 2017. "More Than Just a Pretty Face: Using SAS® Output Delivery System to Create Microsoft Excel Worksheets That Answer Those Difficult Question". Proceedings of the SAS Global Forum, Orlando, FL. http://support.sas.com/resources/papers/proceedings17/SAS0710-2017.pdf

Parker, Chevell. 2017. "ODS HTML5 in the Fourth Maintenance Release for SAS® 9.4". Proceedings of the South Central SAS Users Group, Dallas, TX.

Parker, Chevell. 2018. "Insights from a SAS Technical Support Guy: A Deep Dive into the SAS® ODS Excel Destination". Proceedings of the SAS Global Forum, Denver, CO. https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2174-2018.pdf

Rhodes, Dianne Louise. 2002. "Developing and Maintaining a Tips Database: A Practical Approach to Programming Standards, Style Sheets and Peer Reviews". Proceedings of the SAS Users Group International, Orlando, FL. http://www2.sas.com/proceedings/sugi27/p235-27.pdf

Rhodes, Dianne Louise. 2004. "Programming Standards, Style Sheets, and Peer Reviews: A Practical Guide". Proceedings of the SAS Users Group International, Montreal, Quebec, CANADA. http://www2.sas.com/proceedings/sugi29/135-29.pdf

Sandlin, Robin M.. 2015. "Phantom of the ODS ? How to run cascading compute blocks off of common variables in the data set for complex tasks". Proceedings of the PharmaSUG, Orlando, FL. http://www.lexjansen.com/pharmasug/2015/TT/PharmaSUG-2015-TT02.pdf

Smith, Kevin. 2011. "Unveiling the Power of Cascading Style Sheets (CSS) in ODS". Proceedings of the SAS Global Forum, Las Vegas, NV. http://support.sas.com/resources/papers/proceedings11/297-2011.pdf

Smith, Kevin. 2013. "Cascading Style Sheets: Breaking Out of the Box of ODS Styles". Proceedings of the SAS Global Forum, San Francisco, CA. http://support.sas.com/resources/papers/proceedings13/365-2013.pdf

Yeh, Shi-Tao. 2000. "Using Style Sheets to Format SAS Output Web Page Layout". Proceedings of the Northeast SAS Users Group, Philadelphia, PA. http://www.lexjansen.com/nesug/nesug00/ps/ps7023.pdf

Zender, Cynthia. 2009. "CSSSTYLE: Stylish Output with ODS and SAS® 9.2". Proceedings of the SAS Global Forum, National Harbor, MD. http://support.sas.com/resources/papers/proceedings09/014-2009.pdf

ACKNOWLEDGMENTS

The authors wish to acknowledge Bari Lawhorn, Kevin Smith, Jane Eslinger, and Chevell Parker of SAS who work tirelessly to improve and facilitate the use of reporting with SAS.

CONTACT INFORMATION

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

troymartinhughes@gmail.com
Louise_hadden@abtassoc.com

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.