Paper 221-2010

Personalized Web Reports in Style: Tweaking SAS® ODS, Tagsets, and CSS to Get the Output Right

I. Liebhardt, European Patent Office, The Hague, Netherlands

ABSTRACT

This paper aims to present a solution for a scenario wherein a vast amount of less experienced users (e.g., heads of sections) are to be provided with Web reports that are personalized in the sense that each user automatically obtains data for those business units lying within his or her responsibility. The layout of these reports, on the other hand, should be common to all users and fully aligned to the styling of the remainder of the organization's intranet. We obtain this type of report by using SAS® Stored Processes, by overwriting parts of the predefined SAS® tag sets, and by extensively adding identifiers and classes to the HTML output. The optical appearance is thereafter defined by means of cascaded style sheets.

INTRODUCTION

SAS® software offers a vast amount of alternative products suitable for publishing reports to the web. SAS® Web Report Studio and SAS/IntrNet® are two examples. SAS® Stored Processes and the SAS® Portal are further examples. Even experienced users might be confronted with difficulties when trying to decide which solution to chose for a particular project. Needless to say that each approach has its own strengths and drawbacks and that the optimal choice heavily depends on the envisaged scenario.

In the scope of this paper, the designers were confronted with the following scenario:

- the report to be published to the organization's intranet is a standard report containing a table and several graphs;
- on demand, the report is to be generated on the fly with data originating from data marts that are designed particularly for this report and that are updated monthly;
- the report is viewed by several hundreds of different users;
- even inexperienced users should be able to easily view and print the report;
- layout and optical appearance are common to all reports, independently of the user viewing the report;
- layout and optical design should seamlessly integrate into the existing intranet and the organization's corporate design should be followed;
- however, the data to be displayed to each of the users should be personalized in the sense that each user automatically obtains data for those business units lying within his or her responsibility.

This is a scenario that could, for example, typically occur for a chain of department stores wherein each store manager has access to a sales report automatically showing the figures of that particular store.

It was for the reason of automatically identifying the user that SAS® Stored Processes have been chosen for the implementation of this type of report. SAS® Stored Processes yield the advantage that the user calling the stored process is easily identified by means of the reserved macro variable _METAUSER. Furthermore, SAS® Stored Processes can conveniently be called by means of a URL locating the stored process inside the SAS® Portal. The latter is very advantageous for the seamless integration into the organization's existing intranet. However, layout and optical appearance of the output of a SAS® Stored Process do normally not integrate very well into an existing intranet. Furthermore, the degree of freedom with respect to style adjustments that can normally be performed to the output of a stored process is far too limited to allow for a seamless integration into an existing intranet.

Especially in security sensitive environments, where it should be assured for privacy protection reasons that each user exclusively views his or her own data, building upon the strengths of SAS® Stored Processes is a real advantage. In any case, the simplicity of identifying users by means of the reserved macro variable _METAUSER in stored processes comes in very handy in this context.

The solution presented in this paper has been introduced to overcome all the above limitations whilst profiting from the advantages of SAS® Stored Processes.

In the following, the reader will find detailed information on how to

- modify an existing SAS® Tagset to make it fit particular needs;
- use the concept of inheritance for SAS® Tagsets;
- create a table wherein the output style can be defined for each line, for each column, and even for each cell individually;
- place headers and navigation items that are in line with an organization’s existing intranet; and
- add several graphs to the output.

Furthermore, a brief introduction into the concepts of CSS will be given and browser incompatibilities will be briefly discussed. The later are one of the major difficulties that need to be overcome in a real-world implementation characterized by an environment with several different web browsers. Our scenario, which is used purely as an example herein, is a reporting for individual patent examiners working for the European Patent Office. After having read this publication, the reader should be able to adapt the methodology described herein to any comparable scenario.

MODIFYING SAS® ODS TAGSETS

To begin with, a suitable SAS® ODS Tagset has to be chosen as the starting point for the definition of the desired output. As it is intended to define all details of the optical appearance at a later stage by means of CSS, the choice of a very simple HTML-based tagset is normally most advantageous.

In the present case, we are starting from the Compact Hypertext Markup Language (CHTML), which is the best starting point ensuring compatibility. We use the concept of inheritance to simply take over most of the definitions from the original CHTML tagset delivered by SAS®. However, we redefine the events ‘doc’, ‘doc_head’, ‘image’, ‘doc_title’, and ‘pagebreak’ according to our needs. Firstly, we define the document type as HTML 4.01. Then we insert a link to our style sheet, wherein we will define the report’s optical appearance at a later stage. Furthermore we want the title field of the browser window to display the text ‘Examiner Production & Stock Report’. All this is shown in the code excerpt below, using the TEMPLATE procedure.

```
ODS PATH(PREPEND) work.templat(update);

PROC TEMPLATE;
define tagset tagsets.dmbrtags;
parent=tagsets.chtml;

define event doc;
start:
    put '<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">' CR;
    put CR CR;
    put "<html>" CR;
finish:
    put "</body>" CR;
    put "</html>" CR;
end;

define event doc_head;
start:
    put "<head>" CR;
    put '<link rel="stylesheet" href=" [insert reference here] " type="text/css"
/>' CR;
    put VALUE CR;
finish:
    put "</head>" CR;
    put "<body>" CR;
end;

define event image;
    putq '<div class=' alt;
    put ">" CR;
    put '<h3 class="image_heading">' alt;
    put '</h3>' CR;
    put '<img';
    putq ' alt=' alt;
```

```
        put ' usemap="#' @CLIENTMAP;
        put ' usemap="#' NAME / if !exists(@CLIENTMAP);
        put '"' / if any(@CLIENTMAP , NAME);
        putq " src=" URL ">" CR ;
        put '</div>' CR;
    end;

    define event doc_title;
        put "<title>";
        put "Examiner Production &amp; Stock Report";
        put "</title>" CR;
    end;

    define event pagebreak;
    end;

    end;
    RUN;
```

It is of utmost importance to include the above code into the stored process before the appearance of the code line '%stpBegin;'. Care has to be taken when editing the stored process's code in SAS® Enterprise Guide to ensure that the automatic inclusion of '%stpBegin;' and '%stpEnd;' is disabled. Finally, we set the ODS output destination to the tagset we defined above by including the line of code '%let _ODSDEST = tagsets.dmbrtags;' above the line '%stpBegin;'. The newly defined tagset can now be used for our output.

PREPARING THE OUTPUT

We can now already use the tagset defined above. However, page titles, headers, table elements, and graphs still have to be prepared in a CSS friendly manner prior to the output. This is explained in the following.

PREPARING THE PAGE TITLES

The design envisaged for the topmost part of the report is a bar stretching over the whole width of the browser window and displaying a logo, the name of the report, the name of the user viewing the report, the organizational unit of the user viewing the report, and the reporting period. The optical appearance is, of course, defined by means of CSS at a later stage. The data to be displayed is, on the other hand, obtained by means of the following code that follows the line '%stpBegin;' in the stored process. To begin with, the user's organizational unit and the user's name are obtained from a table, wherein the macro variable 'USER_PERS_NO' is used as a filter. The latter variable is obtained directly from the reserved macro variable '_METAUSER'. The reporting month is obtained from another table and transformed into a text string indicating the reporting period. Finally the titles are output. All output is provided with classes that are used later on to define the styles in CSS.

```
    PROC SQL NOPRINT;
        SELECT DISTINCT
            TDW002_EXAM_DIM.ORG_KEY FORMAT=11.,
            ('<div class="dir_code"> Directorate ' || TDW008_ORG_DIM.ORG_CODE ||
    '</div>'),
            ('<div class="exam_name">' || TRIM(TDW002_EXAM_DIM.EXAMINER_LASTNAME) ||
    ", " || TDW002_EXAM_DIM.EXAMINER_FIRSTNAME || '</div>') into :ORG_KEY, :ORG_CODE,
    :NAME_TITLE
            FROM DMDMBR.TDW002_EXAM_DIM AS TDW002_EXAM_DIM
            INNER JOIN DMDMBR.TDW008_ORG_DIM AS TDW008_ORG_DIM ON
    (TDW002_EXAM_DIM.ORG_KEY = TDW008_ORG_DIM.ORG_KEY)
            WHERE TDW002_EXAM_DIM.EXAMINER_PERS_NO = "&USER_PERS_NO" AND
    TDW002_EXAM_DIM.ORG_KEY NE 1;
        QUIT;

    PROC SQL NOPRINT;
        SELECT
            REPORTING_MONTH,
            MONTH(REPORTING_MONTH),
            YEAR(REPORTING_MONTH),
            YEAR(REPORTING_MONTH)-1
            INTO:REPORTING_MONTH, :MONTH_STR, :YEAR_STR, :PREV_YEAR_STR
            FROM METADATA.TDW910_REPORTING_MONTH;
        RUN;

    %MACRO preparetitles;
        %if &MONTH_STR=1 %then %do;
```

```
          %let PERIOD=Jan &YEAR_STR ; %end;
          %else %if &MONTH_STR=2 %then %do;
            %let PERIOD=<div class="period">Jan - Feb &YEAR_STR </div>; %end;
          %else %if &MONTH_STR=3 %then %do;
            %let PERIOD=<div class="period">Jan - Mar &YEAR_STR </div>; %end;

      [ . . . ]

          %else %do;
            %let PERIOD=<div class="period">Jan - Dec &YEAR_STR </div>; %end;

%MEND preparetitles;

%preparetitles

TITLE;
TITLE1 '<div class="maintitle"> Examiner Production & Stock Report </div>';
TITLE2 &ORG_CODE;
TITLE3 &NAME_TITLE;
TITLE4 &PERIOD;
```

DEFINING THE TABLE'S LINES

The primary aim concerning the output table is to be able to set any styling separately for each of the lines forming the table. Since this paper focuses on the visualization of data, the details for obtaining the data table itself are not within the scope of this paper. Suffice it to say that the data to be displayed in this report is present in a table and is organized as follows:

- the table consists of 15 columns, 12 for all months of the year, one column for the accumulated year-to-date value, and one column for the previous year's value;
- each line is provided with an item name, which indicates the measure kept in that particular line; and
- numeric data is displayed for all months up to - and including - the reporting month, all remaining months have empty values.

For displaying the final table, it is first of all necessary to attribute a 'readable' table line heading to each of the rather abstract item names. Furthermore, the order in which the lines should be displayed is defined. As a mater of convenience, the number of digits to be displayed after the decimal point is defined here as well. All this is obtained by creating a rather simple table with four columns. The first column contains the item name and the second one contains the table line heading describing the line's contents to the user. The third line is a numeric value setting the line's position inside the table and the last column is a numeric value defining the amount of decimals to be displayed after the decimal point. Since the table's structure is rather simple, only a couple of lines of the corresponding SQL code are exemplarily shown here.

```
VALUES ('EXAM_HEADING', 'Examination', 7, 0)
VALUES ('A_EXA_AWAFIX', 'Awaiting First Comm.', 8, 0)
VALUES ('A_EXA_AWAFUX', 'Awaiting Further Comm.', 9, 0)
VALUES ('A_EXA_ACTIVESTOCK', 'Active Stock (1st & Further)', 10, 0)
```

PREPARING THE OUTPUT TABLE

The table 'METAINFO' created as described in the above subsection can now be joined with the data table to prepare the final output. For doing this, a left join is used and each of the 15 columns is treated separately during the join. This is done to provide an appropriate class attribute to each and every cell of our output table, so that formatting can then be applied by means of CSS for each value individually. Furthermore, each output value is isolated into an HTML division by means of the HTML codes '<div>' and '</div>' and the aforementioned class attribute is provided to these divisions. In this context, it is helpful to build upon a consistent naming methodology. What has been used as an item name in the data table discussed above is now applied as the name for the class attribute. This consistency eases the effort to be invested for the CSS definitions at a later stage. Finally, all empty values are converted into the HTML code '&nbsp', which stands for non-breaking space. This is done to ensure the correct display of tables with empty values independently of the used browser. The SQL code responsible for this final preparation of the output is shown below. Although the code might look overly complex on first sight, the actual structure underlying all conversions is identical for al the columns, making it a lot easier than what would be expected on first sight.

```
PROC SQL NOPRINT;
CREATE TABLE WORK.Q_DATA AS SELECT
```

```
    ('<div class="item_name_' || LOWCASE(COMPRESS(METAINFO.ITEM_NAME)) || '">' ||
TRIM(HEADER) || '</div>') AS HEADER,
    (CASE
          WHEN M_1 is missing THEN ('<div class="' ||
LOWCASE(COMPRESS(METAINFO.ITEM_NAME)) || '"> &nbsp </div>')
          WHEN DECIMALS=1 THEN ('<div class="' ||
LOWCASE(COMPRESS(METAINFO.ITEM_NAME)) || '">' || TRIM(PUT(M_1,6.1)) || '</div>')
          WHEN DECIMALS=2 THEN ('<div class="' ||
LOWCASE(COMPRESS(METAINFO.ITEM_NAME)) || '">' || TRIM(PUT(M_1,6.2)) || '</div>')
          ELSE ('<div class="' || LOWCASE(COMPRESS(METAINFO.ITEM_NAME)) || '">'
|| TRIM(PUT(M_1,6.)) || '</div>')
          END) AS M_1,

[ . . . ]

    (CASE
          WHEN M_PY is missing THEN ('<div class="' ||
LOWCASE(COMPRESS(METAINFO.ITEM_NAME)) || '"> &nbsp </div>')
          WHEN DECIMALS=1 THEN ('<div class="' ||
LOWCASE(COMPRESS(METAINFO.ITEM_NAME)) || '">' || TRIM(PUT(M_PY,6.1)) || '</div>')
          WHEN DECIMALS=2 THEN ('<div class="' ||
LOWCASE(COMPRESS(METAINFO.ITEM_NAME)) || '">' || TRIM(PUT(M_PY,6.2)) || '</div>')
          ELSE ('<div class="' || LOWCASE(COMPRESS(METAINFO.ITEM_NAME)) || '">'
|| TRIM(PUT(M_PY,6.)) || '</div>')
          END) AS M_PY
FROM WORK.METAINFO AS METAINFO
LEFT JOIN WORK.FINAL_EXAM0 AS FINAL_EXAM0
ON (METAINFO.ITEM_NAME = FINAL_EXAM0.ITEM_NAME)
ORDER BY POS;
QUIT;
```

DISPLAYING THE TABLE

After all the above preparations are in place, the only task left to be done is the output of the final table created by the left join as explained in the previous subsection. This is done by means of the rather simple code, using the PRINT procedure as shown below.

```
PROC PRINT DATA = WORK.Q_DATA NOOBS LABEL;
    LABEL HEADER='<div class="top_left_header"> &nbsp </div>'
        M_1='<div class="top_header"> Jan </div>'
        M_2='<div class="top_header"> Feb </div>'
        M_3='<div class="top_header"> Mar </div>'
        M_4='<div class="top_header"> Apr </div>'
        M_5='<div class="top_header"> May </div>'
        M_6='<div class="top_header"> Jun </div>'
        M_7='<div class="top_header"> Jul </div>'
        M_8='<div class="top_header"> Aug </div>'
        M_9='<div class="top_header"> Sep </div>'
        M_10='<div class="top_header"> Oct </div>'
        M_11='<div class="top_header"> Nov </div>'
        M_12='<div class="top_header"> Dec </div>'
        M_CY='<div class="ytd_top_header"> YTD* </div>'
        M_PY='<div class="top_header"> PY* </div>';
RUN;

TITLE;
TITLE1 ;
TITLE2 ;
TITLE3 ;
TITLE4 ;
```

As one can see, this is also the location inside the stored process's code where all the previously used titles are reset because it is not intended to display them again together wit the graphs.

ADDING GRAPHS

The report now contains the header on top of the page and the table containing all numeric values of importance. All that is left to be done is the addition of the four graphs to be displayed together with the table. The data to be displayed in the graphs is already present in a table called 'GRAPH3'. This table contains the evolution of certain measures during the months from January up to, and including, the reporting month. It is beyond the scope of this paper to show how this table is filled with values. All options concerning the graphs can easily be set by means of the 'goptions' command. Thereafter, each of the graphs is displayed by means of the GPLOT procedure as exemplarily shown below for one of the graphs.

```
PROC GPLOT DATA = WORK.GRAPH3 ;
PLOT P_EXA_PROEXA * MONTH PY_EXA_PROEXA * MONTH A_EXA_PROEXA * MONTH  /
        OVERLAY
    VAXIS=AXIS1
    HAXIS=AXIS2
    FRAME
    LEGEND=LEGEND1
    DESCRIPTION='Examination Production (Cumulative)';
RUN;
```

The same can be done for all of the graphs. A further advantage is that this type of report is very flexible as to the number of graphs to be included.

DEFINING THE CASCADED STYLE SHEET

Upon invoking the stored process containing all the necessary code elements explained above, results are already displayed in a web browser. However, these results still lack formatting, which is to be applied now by means of CSS. This last step is explained in this section.

GENERAL ASPECTS OF CSS

Cascaded Style Sheets are an extension to HTML. They are specified in a language for defining the formatting of particular HTML elements. By means of CSS you can, for example, define that all first order headings should have a large font size, a dark blue color, and a font of the type 'Helvetica'. You can furthermore define background colors, wallpapers, absolute or relative positioning, margins and much more. However, one of the most advantageous features of CSS is the separation of the content from the definition of its optical appearance: whilst HTML used to contain the text and data to be displayed interspersed with commands for formatting, CSS enables a logical separation of these two elements. CSS furthermore supports the powerful concepts of cascading and inheritance. A good starting point for learning more about CSS is provided in [1], whilst [2] also contains one chapter extensively dealing with formatting by means of CSS.

IDENTIFIERS AND CLASSES

One characteristic that is reappearing frequently in the code presented above is the assignment of a class in HTML. This is done by adding '<div class=" . . . ">' and '</div>' as some kind of wrapper around all our output elements. Wrapping output elements this way ensures that they can be addressed individually by means of CSS.

In the above code, we created a first order heading containing a division with the class called 'maintitle'. If we now want to define the style of this part of our report, we do this as exemplarily presented in the code below.

```
h1 div.maintitle {
    margin-top: 10px;
    margin-left: 10px;
    margin-right: 10px;
    padding-top: 12px;
    text-align: center;
    height: 65px;
    border-bottom: 1px solid white;
    font-size: 18pt;
    font-weight: normal;
    font-style: normal;
    text-decoration: none;
    color: white;
    font-family: Verdana, Helvetica, Arial, sans-serif;
    background: rgb(51,51,51) url(epo.gif) no-repeat left top; }
```

The above part of the CSS defines margins, sizes, font sizes, background colors, font types and much more. It also includes a company logo in the top left part of the report.

As to the table, we first define a standard styling to be applied to the table in its entirety. The borders normally applied to tables are disabled because a custom style is to be applied to each of the lines individually.

```
table {
    border-collapse: collapse;
    color: black;
    line-height: 1.4;
    font-size: 9pt;
    border-style: none;
    background-color: white;
    width: 60em;
    float: left;
    margin-top: 25px;
    margin-left: 15px;
    margin-right: 15px;
    margin-bottom: 15px;
    text-align: right;
}

td, th {
    border-style: none;
}
```

As we assigned individual classes for many of the table's elements, we are now able to address them individually for the sake of formatting. This is exemplarily shown below for elements of the class 'a_cap_setime'.

```
div.a_cap_setime {
    border-bottom: 1px solid silver;
    text-align: right;
    font-weight: bold;
    padding-right: 2px;
    padding-left: 2px;
}
```

The above can now be done for all the elements forming the table and elements that are to be displayed in the same style can conveniently be grouped together.

Although only classes have been used in the above code, it should also be noted that identifiers may be used as well. Identifiers are used by replacing the above wrapping of the output by the code '<div id=" . . . ">' and '</div>'. In the CSS, these identifiers are thereafter addressed by '#identifier_name'. Although classes and identifiers are nearly identical in nature, it should be borne in mind that identifiers may appear only once in the generated HTML code whilst one single class may have multiple occurrences.

LETTING THE GRAPHS FLOAT

The only task left to be discussed is the positioning of the graphs around the table. In this particular scenario, the graphs are comparatively small and should therefore float around the table. The table is to be located on the left hand of the browser window, directly under the header with its company logo, report name, reporting period and user name. The graphs should then float around the table. In this case, it is the table, and not the graph, that has to be attributed with a float keyword. If we recall the above CSS code for formatting the table, we see that this is achieved by the line 'float: left;' that forms part of the table's formatting.

After this is done, graphs can, for example, be formatted as follows:

```
div.Graph1 {
    background-color: white;
    width: auto;
    text-align: center;
    font-size: 9pt;
    float: left;
    margin: 25px 15px 25px;
}
```

BROWSER (IN)COMPATIBILITIES

One of the problems that will have to be countered in any real-world application that adopts the methodology presented above is the problem of browser incompatibilities. If users of an organization or a company run two or more different types of web browsers, this issue might become a real hassle. The reason for this is the fact that not all browsers support all the commands of CSS. Furthermore some commands might be interpreted in different ways by different browsers. It is therefore important to frequently check all layouts during their creation. Recognizing potential issues with one particular browser at an earlier stage can avoid overly complex corrections at a later stage. A lot of resources on the different interpretation and on the limited support of CSS commands may be found on the web. The detailed tables provided in [3] and [4] are just two of the many examples that may be consulted.

CONCLUSION

The above example demonstrates what SAS® ODS can do if it is combined with formatting by means of Cascaded Style Sheets. The output thus obtained has an impressive optical appearance. Nonetheless, the programming effort is still kept comparatively low. The example presented above can easily be adapted and extended. For example, a separate style sheet for printing may be added. CSS, once unleashed, allows for a huge amount of creativity.

REFERENCES

[1] Mansfield, R.: “CSS Web Design for Dummies”, Wiley Publishing, Inc., 2005, ISBN: 0-7645-8425-1

[2] Harold, E. R.: “XML 1.1 Bible”, 3rd Edition, Wiley Publishing, Inc., 2004, ISBN: 0-7645-4986-3

[3] http://www.quirksmode.org/css/contents.html

[4] http://www.westciv.com/style_master/academy/browser_support

CONTACT INFORMATION

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

Name:	I. Liebhardt
Enterprise:	European Patent Office
Address:	Patentlaan 2
City, State ZIP:	2288 EE Rijswijk, Netherlands
Work Phone:	+31 70 340 4114
E-mail:	iliebhardt@epo.org
Web:	www.epo.org