Paper 012-2008

Powering Your Organization's Web Applications With SAS® Software

Michael A. Raithel, Westat, Rockville MD.

Abstract

SAS developers who do not license SAS/IntrNet software may not think that there is a way they can harness the power of SAS for their organizations' Internet applications. But, there is! SAS can be used as a robust behind-the-scenes engine to create web pages and web content for Internet applications. The powerful data handling and report-creating features of SAS can be used to craft web pages and to load databases with relevant information that is used in web applications.

This paper illustrates five ways that SAS can be integrated into corporate Internet web applications. It shows how SAS can be used as an engine to update web pages with up-to-date drop-down lists based on current values in a database; to create up-to-date links to newly created web report files; to create new web pages from flat file reports; to create new web application pages directly in a web directory using the Output Delivery System; and how to extract data from a DBMS, process it, and then load rows into DBMS tables that are later used to create pages in a web application. The complete programs for each example are included and discussed. After reading this paper, you should be able to adapt any one of these methods so that your own organization can use SAS to power its web applications.

Overview

SAS/IntrNet is a separately-licensed SAS software product that allows organizations to implement web pages that call upon SAS to produce reports and other result sets over the web. Organizations that have SAS/IntrNet can leverage the power of SAS analytics to provide their web users with reports, tables, graphs and downloadable result data sets. SAS/IntrNet application web pages contain special HTML tags that specify which SAS programs are to be executed on the back-end servers, and that pass along user selection criteria such as which items were selected from a drop-down list or were checked in a group of check boxes. SAS/IntrNet, running on back-end servers, accepts the input from web pages, executes the specified SAS programs, processes data, and sends the result set back to the clients' web browsers.

Organizations that do not license SAS/IntrNet software can still use the power of SAS to create content for their Internet and intranet web applications. However, it will not be as dynamic as when using SAS/IntrNet. That is; when SAS/IntrNet is executed, any web pages that are created exist for a brief period of time—only long enough to be sent back to the individual who requested them via the web. Then, they are gone. Using SAS behind the scenes to power web applications calls for the creation of static content. So, static web pages are created and stored in an organization's web directories. These static web pages will be available for all users who invoke them via links in the web applications. SAS can also be used to create descriptive statistics that are stored in tables in a database. These tables may then be used by web applications to provide descriptive statistics to web application users.

The key to this process is to execute SAS programs in batch on a back-end server to create web pages or SAS-analyzed data that is subsequently used in a web application. The SAS programs can be scheduled on a Windows server, a UNIX server, a Linux server, or any other type of server, to execute on a cyclic basis. Often, such back-end SAS programs only need to execute once a night, to update web pages, or database tables, with information from "yesterday". However, the back-end programs can be executed whenever it is practical to do so.

Here is a general overview of the process:

- During the day, database tables (SQL Server, Oracle, etc.) or SAS data sets are updated by transactional processing.
- After midnight, SAS batch jobs execute and process data from the updated database tables or SAS data sets:
  - Web application HTML pages are *updated*
  - Web application HTML pages are *created*
  - Web application database tables are updated
- When users access the web application in the morning, they have access to the newly updated HTML pages and/or the freshly updated database tables.

Under this scenario, web programmers create the original web pages, which include selection lists, check boxes, radio buttons, etc. SAS programmers create batch programs that execute nightly and update the selection lists, check boxes, radio buttons, etc. in the web pages with values currently found in the data. This scenario allows organizations to make the best use of staff and resources. Web programming staff can concentrate on the functionality, and the look and feel of web applications. SAS programmers can concentrate on manipulating data to create content for the web applications. So, web programmers do not have to worry about the intricacies of correctly processing data, and SAS programmers do not have to worry about the intricacies of correctly bringing information to the web.

As mentioned earlier, the SAS programs are generally scheduled to run at off hours, such as the early hours of the morning, to create content. SAS programmers can make use of schedulers such as the CRON facility on UNIX, the Windows Scheduler on Windows

servers, and the CA-7 scheduler on z/OS, to schedule SAS batch programs. On Windows, a .bat file is needed to run scheduled SAS programs in batch. Here is an example:

```
"C:\Program Files\SAS Institute\SAS\V9\SAS.EXE" -icon -noterminal -nosplash
-noxwait -noxsync
-CONFIG "C:\Program Files\SAS Institute\SAS\V8\Sasv9.cfg"
-SYSIN "Q:\programs\WebApp_Driver_Program.sas"
-LOG "Q:\programs\WebApp_Driver_Program.log"
```

That .bat file executes the WebApp_Driver_Program SAS "shell" program that %INCLUDES all of the SAS programs that must run:

```
/*************************************************************/
/* WebApp_Driver_Program for Corporate Internet Applications */
/*************************************************************/

*****************************************************;
* INCLUDE web page update programs.                 *;
*****************************************************;
%include 'C:\My SAS Programs\LOADHTML.sas';
%include 'C:\My SAS Programs\GETFILES.sas';
%include 'C:\My SAS Programs\FILE2HTM.sas';
%include 'C:\My SAS Programs\IDENTIFY.sas';
%include 'C:\My SAS Programs\ODSHTML.sas';
%include 'C:\My SAS Programs\Process and Load Server Tables.sas';

*****************************************************;
* Execute web page update programs.                 *;
*****************************************************;
%LOADHTML(C:\TEMP,OrionStarReports.htm,REPORTYEAR,orsales,YEAR);
%GETFILES(C:\bigwebapp\InputDir,SurefootShoeReports.html,C:\inetpub\bigwebapp);
%IDENTIFY(D:\Reportlib,C:\inetpub\bigwebapp);
%ODSHTML(C:\inetpub\bigwebapp);
```

The rest of this paper is devoted to five examples of how to run SAS on back-end servers to create content for web applications. All of the examples illustrate SAS running in a Windows server environment. However, they can easily be modified to run on other operating systems. These examples are presented as proof-of-concept programs that can serve as your own jumping-off point for creating static content for your organization's web applications.

Example 1 – Updating Static HTML Page Selection Lists

In this example, a web programmer has created a web page that contains two selection lists. That web page is an integral part of a web application. Users can navigate to this web page, based on choices made while using the web application. The web page exists in a directory on a web server, along with other static web pages. A nightly SAS program updates the web page by executing the LOADHTML SAS macro two times. (The LOADHTML SAS macro can be found in Appendix A at the end of this paper).

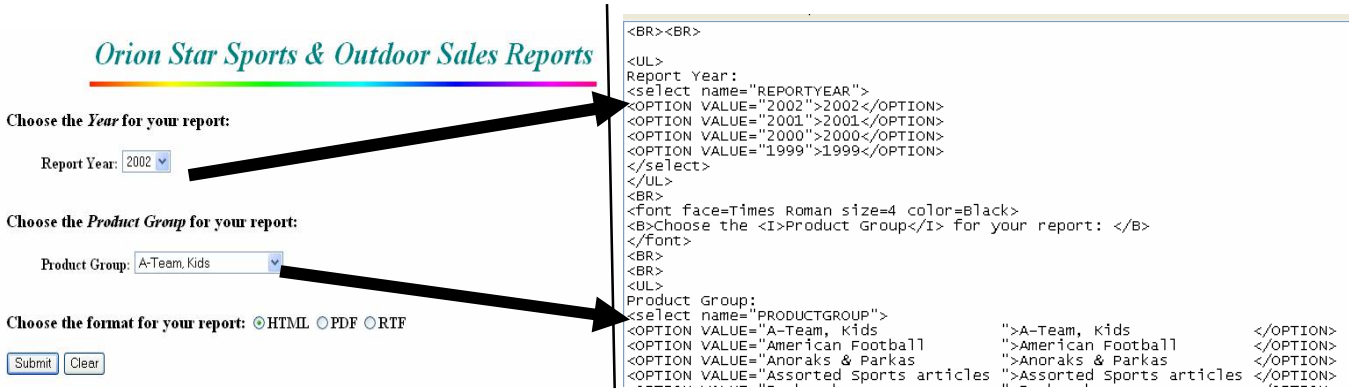

Figure 1 – The web page as displayed in a browser, and as displayed in Wordpad as an HTML file.

Figure 1, above, shows the web page as it is displayed in a web browser and as it looks as an HTML file displayed in Wordpad. The arrows show the relationship between the selection lists for Report Year and Product Group as displayed in a web browser and as they exist in the HTML file. It is these two drop-down lists that are updated by two different executions of the LOADHTML SAS macro.

Before the LOADHTML macro is executed, the SQL Server data base that contains tables with the information needed for the Report Year and Product Group selection lists is allocated via a LIBNAME statement. It is important to use ***SQLLIB*** as the Libref, because the LOADHTML SAS macro uses that Libref when referring to tables in the database that is allocated.

The LOADHTML SAS macro uses the following five parameters:

- **DIRNAME** – The directory name where the HTML page file can be found
- **HTMLPAGE** – The name of the HTML page that is to be modified
- **SELNAME** – The name of the selection list (in the "<select name" HTML tag) that is to be updated
- **TABLENAME** – The name of the SQL Server table that contains information needed to update the selection list
- **VARNAME** – The name of the variable in the SQL Server table that will be used to build HTML "<option value" tags. One of these tags will be created for every unique value of the variable specified for VARNAME.

The LOADHTML SAS macro performs the following steps:

1. **Read SQL Server database using SAS**. LOADHTML uses SAS/Access to Relational Databases to access SQL Server database tables. Note that the data could have been in Oracle, Sybase, or in SAS data sets.

2. **Summarize dates and summarize product groups**. LOADHTML executes PROC SQL to read the ORSALES SQL Server table and create a SAS data set. In this example, it is executed two times. The first time, LOADHTML is executed for the REPORTYEAR selection list to get the *unique* values of YEAR found in the ORASALES table. It creates the YEAR SAS data set. The second time, LOADHTML is executed for the PRODUCTGROUP selection list, which contains *unique* values of PRODUCT_GROUP found in the ORASALES table. It creates the PRODUCT_GROUP SAS data set.

   Note that new values of either YEAR or PRODUCT_GROUP could have been added to the ORASALES table during transactional processing during the day. That is why this program is run nightly—to capture the unique values that now exist in the table and populate the selection lists with the latest values.

3. **Create SAS data set with dates/product groups surrounded by HTML tags**. LOADHTML continues by reading the new SAS data set that it just created (YEAR and then PRODUCTGROUP), and creates a new variable named STRING1. STRING1 has the name/value pairs for an HTML OPTION VALUE tag. Each unique observation in one of the SAS data sets (YEARS and then PRODUCT_GROUP) is turned into a syntactically correct HTML "OPTION VALUE" tag.

4. **Read HTML web page file**. LOADHTML reads the HTML web page file as if it were any flat file containing raw data. As it reads each line, it writes that line out to a temporary new HTML file. It will do so until it gets to the drop-down list identified by the "SELECT NAME" tag in the HTML web page file, and identified by the SELNAME parameter in the LOADHTML macro.

5. **Delete current drop-down list entries**. The SAS program deletes each of the lines from the *old* HTML web page file that contain "OPTION VALUE" tags for the selection list named in the SELNAME parameter. These lines are deleted because they are the old values and may not represent the values that currently reside in the SQL Server tables that were updated during the day, or by subsequent batch processing run at night.

6. **Add newly created "OPTION VALUE" tags.** LOADHTML updates the drop-down lists by inserting observations from the previously generated SAS data set. As mentioned earlier, these observations are simply "OPTION VALUE" HTML tags that contain the latest values of, first YEAR, and then PRODUCTGROUP that are found in the SQL Server database. The new "OPTION VALUE" tags are written to the new HTML page.

7. **Switch the new and the old HTML web page files**. At this point, all of the lines from the old HTML page file have been written to the new HTML page file, except the old "OPTION VALUE" lines (which were deleted). The new "OPTION VALUE" tags have been written to the new HTML page file in their place. The old HTML page is now obsolete, so it is deleted. The new HTML page is renamed so that it has the name of the original, old HTML page. Consequently, the web page file that the SAS program was writing to in Steps #4 – 6 is renamed to the name of the original web page file read in.

In Steps #4 - #6, above, SAS reads the HTML file (the web page) as if it would any other flat file. It reads it line-by-line, writing each line out to the temporary HTML file. When it gets to the lines in the file that contain the old drop-down list content, the program deletes those lines. Then, it opens the relevant SAS data set and inserts the new lines that were generated in the first three steps of the program. As stated, all of this is done once for the REPORTYEAR selection list, and then again for the PRODUCTGROUP selection list in two consecutive executions of the LOADHTML SAS macro.

Example 2 – Updating Links to Web Pages That Were Created by Other Applications

This example capitalizes on SAS's ability to process server directory information and SAS's ability to process web page files as data files. It uses the GETFILES SAS macro. That macro reads a directory on a web server that contains HTML files created by some other application. The HTML files are usually report files that are to be made available to users of a web application via a web page that contains HREF web tags. The GETFILES SAS macro obtains the full-path file names of the web reports and updates the web page containing the HREF web tags. (The GETFILES SAS macro can be found in Appendix B at the end of this paper).

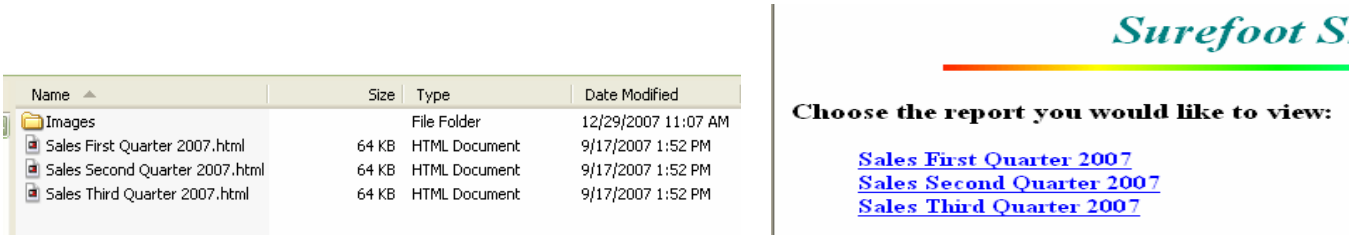

Figure 2 – Relationship between directory of HTML report files and the HREF links in the production web page.

Figure 2, above, illustrates the relationship between the HTML report files in the web report directory and the HREF links in the production web page. The GETFILES SAS macro creates a direct link to each report file in a specified directory and puts those links in the specified production HTML web page. Consequently, users clicking on a link will have the corresponding HTML report file surfaced in their web browsers.

The GETFILES SAS macro uses the following three parameters:

- **HTMLDIR** – The name of the directory containing the HTML report files
- **HTMLFILE** – The name of the HTML page that will have its selection list modified
- **OUTPUTDIR** – The directory containing the HTML file (HTMLFILE) that is to be updated

Here are the details of how the GETFILES SAS macro works:

1. **Read directory of HTML report files**. GETFILES first reads the directory on the web server that contains the HTML report files. The name of that directory is passed to the GETFILES macro via the HTMLDIR macro parameter.

2. **Create file with HTML tags in them**. GETFILES creates an observation for each file it finds in the web server directory. The observations are created in the form of HTML "A HREF" tags. Each HREF tag points to its corresponding file in the web directory. Note that the name of the HTML file becomes the name that is displayed in the HREF tag. So, the names of the files in the HTML directory should be made to be meaningful.

3. **Read HTML web page file**. GETFILES reads the HTML web page file as if it were a flat file. As it reads each line, it writes that line out to a temporary new HTML file. It will do so until it gets to the "A HREF" tags in the HTML web page file.

4. **Delete current "A HREF" entries**. GETFILES deletes each of the lines from the original HTML web page file that contains "A HREF" tags. This ensures that *old* values are no longer in the *new* web page.

5. **Add newly created "A HREF" tags.** GETFILES updates the new web page by inserting observations from the new SAS data set that contains the latest "A HREF" tags. Consequently, the new web page file now has links to the new, or updated, web pages in the HTML directory.

6. **Switch the new and the old HTML web page files**. After the final line from the old HTML page file is written to the new HTML page file, the old HTML page is no longer needed; so it is deleted. The new HTML page is renamed and given the name of the original, old HTML page.

Example 3 – Create HTML Web Report Files from Report Flat Files

In this example, an application within the organization has already created a directory full of report files on a data server. The report files are basic flat files that we would like to make available to clients via an existing web application. Two SAS macros are used to perform this task: IDENTIFY and FILE2HTM. Both macros are executed by a driver program that %INCLUDEs them in proper order and then invokes the IDENTIFY SAS macro to perform the task.

The IDENTIFY SAS macro reads a directory of report files and creates an invocation of the FILE2HTM SAS program for each entry in the directory. The FILE2HTM SAS macro reads a report file, adds header/footer elements to the file, and writes it out as HTML file to a given web directory. Thus, the IDENTIFY and FILE2HTM macros populate a web directory full of HTML report files that are created

from a server directory of non-HTML report files. (The driver program, and the IDENTIFY and FILE2HTM SAS macros can be found in Appendix C at the end of this paper).

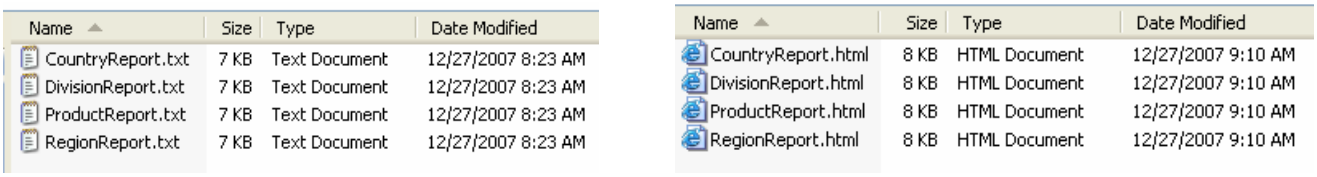

Figure 3 – Report text file in a report directory and report HTML file in a web directory.

Figure 3, above, shows the relationship between a flat file text report and the HTML report. The IDENTIFY and FILE2HTM SAS macros act together to read flat file text reports in a directory on any organizational server and turn them into HTML report files in a web directory.

The IDENTIFY SAS macro does the following:

1. **Process a directory that contains flat-file report files**. IDENTIFY first processes a directory of flat-file reports by inputting each separate report file name into a SAS DATA step piped in via a system "dir" command. Each flat file name is then formatted into a macro call to the FILE2HTM SAS macro. The macro calls are stored in a temporary file.

2. **Invoke the FILE2HTM SAS macro**. After all report flat file names have been read, formatted into separate calls to the FILE2HTM macro, and then stored in a temporary file, the IDENTIFY macro %INCLUDEs the temporary file. This has the effect of executing each macro call to the FILE2HTM SAS macro—one for each flat file that was in the original report directory. Consequently, each report file is read by FILE2HTM, processed, and then written to the target web directory as an HTML report file.

The FILE2HTM SAS macro does the following:

1. **Create a new HTML file and add header tags**. FILE2HTM first gets the name of the report flat file it will be processing. Then, it writes HTML header information to a new HTML file with the same name as the report file. The new HTML file is written to a web directory.

2. **Process the report file**. FILE2HTM reads the report flat file line-by-line as if it were a data file. It writes each line it reads out to the new HTML file. Whenever it finds a new page—represented by "0C"x, it writes an image HTML tag to the new HTML file. That image tag is a reference to a colored horizontal bar that nicely separates logical pages in the report.

3. **Add HTML footer tags**. When all lines from the report flat file have been read into the DATA step and then written to the new HTML file, HTML footer tags are written to that file. Once this is done, the report file in the data directory has been fully converted to an HTML report file in a web directory.

When using the IDENTIFY macro, you may have to change the lengths of the OUTLINE and BIGLINE variables; depending upon the length of your directory and file names. You will need to change the FILENAME statement that pipes in the names of the report files if you move this to another computing platform, such as UNIX. In the FILE2HTM macro, you might have to change the length of the STRING variable, and the LRECL of the HTMLOUT FILENAME, depending upon the width of your report files.

Example 4

This example uses the SAS Output Delivery System (ODS) to create reports that are displayed in an HTML frame that includes a table of contents. This is a classic SAS ODS application! The only twist is that the newly generated HTML files are written *directly* to a web directory after the reports are produced. Once the new HTML report, directory and frame files are created, other application web pages can link to them, and they will look and act as if they are an integral part of the web application.

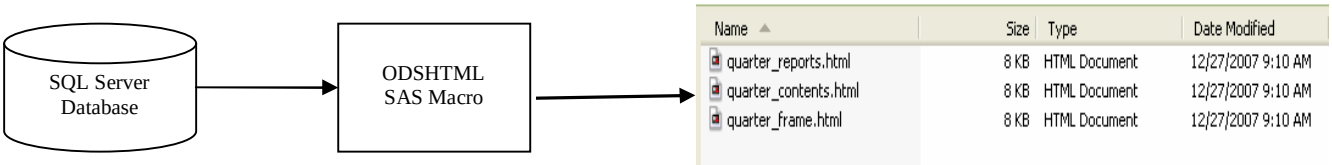

Figure 4 – The ODSHTML SAS Macro reads data directly from tables in a SQL Server database and creates web pages in a web directory on a web server.

This example uses the ODSHTML SAS macro. That macro can be found in Appendix D at the end of this paper. Here is how the ODSHTML SAS macro works:

1. **Read SQL Server database using SAS**. The ODSHTML macro starts by reading data from tables in a SQL Server database. It creates an extract SAS data set with variables based upon the specific needs of the reports it is going to produce. In the example program, it extracts data from a single table and orders the data by several columns. Those columns represent the sort order that clients expect the reports to be created with.

2. **Allocate new HTML files using ODS**. SAS ODS statements are used to allocate body, contents, and frame pages for the reports. These new web page files will be written directly into a web directory on a web server. So, the program that creates them must have the proper system permissions to write to the web directory.

3. **Create new HTML report files using ODS**. The ODSHTML macro reads the extracted table and creates a series of reports. Those reports are written to the body HTML file, and have entries pointing to them created in the directory HTML file.

You can have any number of SAS procedures in the body of the ODSHTML macro to produce any number of reports for the new HTML frame. You can execute the ODSHTML macro once for each frame of reports that you want to produce in the directory of HTML pages on your web server.

Example 5

This final example uses SAS/Access to Relational Databases to both read and write to SQL Server tables. In this scenario, we have a .NET web application that creates dynamic web reports based on information found in SQL Server tables. SAS is used to read the SQL Server database and use its analytic capabilities to create descriptive statistics. Those descriptive statistics are stored in new SQL Server tables so that the .NET web application can surface them to clients. This eliminates the need for the .NET application to use its less sophisticated computational power to derive descriptive statistics.

So, SAS is doing the work of pre-processing the data. It is reading SQL Server tables, and creating descriptive statistics for specific variables, based on criteria that you program. Then, the SAS observations are written into rows of a SQL Server table, so that updated statistics for the data exist in the database. When the .NET web application executes at a future time and retrieves data from the SQL Server tables, it accesses the descriptive statistics previously created by the SAS program. (The example program can be found in Appendix E).

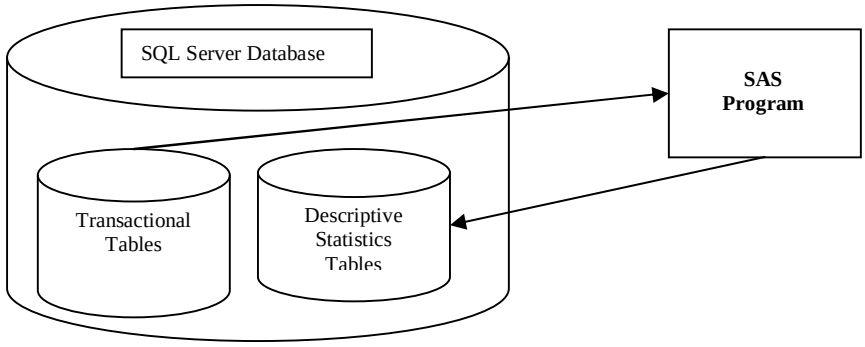

Figure 5 – Diagram of how the SAS program reads tables with transactional data and updates tables with descriptive statistics based on that data.

Here is how this scenario works:

1. **Read SQL Server database using SAS**. A LIBNAME statement allocates the SQL Server database that provides the data that must be processed. SAS/Access to Relational Databases is used to read rows from production SQL Server tables in that database.
2. **Create a table of summary statistics for YEAR**. In the example program, a new table, YEARS_TOTALS, is created from data found in the ORSALES table. For each distinct YEAR, multiple descriptive statistics are calculated for the QUANTITY, PROFIT, and TOTAL_RETAIL_PRICE variables. Each distinct YEAR and the descriptive statistics for the three afore-mentioned variables are written as a row in the new YEARS_TOTALS table.
3. **Create a table of summary statistics for PRODUCT_GROUP**. A second new table, PRODUCT_GROUP_TOTALS, is also created from data found in the ORSALES table. For each distinct PRODUCT_GROUP, multiple descriptive statistics are calculated for the QUANTITY, PROFIT, and TOTAL_RETAIL_PRICE variables. Each distinct PRODUCT_GROUP and the attendant descriptive statistics are written as a row in the new PRODUCT_GROUP_TOTALS table.

Once this SAS program has finished executing, the .NET web application can surface the newly computed descriptive statistics in both the YEARS_TOTALS and the PRODUCT_GROUP_TOTALS tables via SQL calls to those SQL Server tables.

Conclusions

This paper illustrated five ways that you can integrate SAS into corporate Internet web applications without the use of the SAS/IntrNet product. It showed how SAS can be used as an engine to update web pages with up-to-date drop-down lists based on current values in a database; to create up-to-date links to newly created web report files; to create new web pages from flat file reports; to create new web application pages directly in a web directory using the Output Delivery System; and how to extract data from a database, process it, and then load rows into database tables that are later used to create pages in a web application. The five examples were discussed in detail, and example programs were included as appendices to this paper. Now that you have read this paper, you should be able to adapt any one of these methods to power your organization's web applications with SAS software.

Disclaimer

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

References

SAS Institute Inc. 2007. **SAS OnlineDoc® 9.1.3**. Cary, NC: SAS Institute Inc.

Libeg, Linda. (2007). Functionsize: Process Your External Files. Proceedings of the SAS Global Forum 2007 Conference.

Acknowledgements

I would like to thank SUGI 29 Conference Chair Duke Owen for his review and comments on the draft of this paper.

Contact Information

Please feel free to contact me if you have any questions or comments about this paper. You may reach me at:

Michael A. Raithel
Westat
1650 Research Boulevard
Room RW4521
Rockville, Maryland 20850

michaelraithel@westat.com

Appendix A – The LOADHTML SAS Macro

```
*************************************************************************;
* Program:  LOADHTML.sas                                                *;
* This program processes SQL Server tables, finds distinct variable values, then *;
* reads an HTML page and loads those values into the drop-down variable list in  *;
* the page, so it is updated with current values from the SQL Server database.   *;
*                                                                       *;
* This macro accepts five parameters:                                   *;
* DIRNAME - The directory name where the .html page can be found.       *;
* HTMLPAGE - The name of the .html page that is to be modified.         *;
* SELNAME - The name of the selection list in the .html file "<select name" tag  *;
* TABLENAME - The name of the SQL Server table to access.               *;
* VARNAME - The name of the variable in a SQL Server table to build a .html file *;
*           "<option value" tags for.  It builds on tag per unique value.       *;
*************************************************************************;

    /*************************************************************************/
    /*                              Section I                                */
    /*************************************************************************/
    /* This section allocates the SQL Server database.  Note that you must use*/
    /* SQLLIB as the Libref for the SQL Server database because the LOADHTML  */
    /* macro is expecting it for its database table reads.                    */
    /*************************************************************************/

*********************************************************;
* Allocate the SQL Server database.                     *;
*********************************************************;
libname sqllib oledb provider=sqlprov datasource=SQLPRD1
        user=joeuser properties=("Initial Catalog"=sastest)
               prompt=yes schema=dbo;

    /*************************************************************************/
    /*                              Section II                               */
    /*************************************************************************/
    /* This section reads front-end HTML pages and updates the selection lists*/
       /* using the LOADHTML SAS macro.                                      */
    /*************************************************************************/

%MACRO LOADHTML(DIRNAME,HTMLPAGE,SELNAME,TABLENAME,VARNAME);

*********************************************************;
* Get unique values of YEAR from SQL Server database    *;
*********************************************************;
proc sql;
        create table work.&VARNAME as
        select distinct &VARNAME from sqllib.&TABLENAME
        order by &VARNAME;
quit;

*************************************************************;
* Process the PRODUCTGROUP file and create new variables that *;
* will be used to create the updated selection list.          *;
*************************************************************;
data &VARNAME(keep=&VARNAME string1);
set &VARNAME;

        retain optval '<OPTION VALUE="'
               endit  '">'
                  unoptval '</OPTION>'
               ;

        length string1 $80;

        string1 = optval ||  left(trim(&VARNAME)) || endit || left(trim(&VARNAME)) || unoptval;

run;

proc sort data=&VARNAME;
     by &VARNAME;
run;

*****************************************;
* process the old HTML file, creating a new *;
* one with updated Selections.            *;
*****************************************;
```

```
data temp;

retain foundselect 0;

infile "&DIRNAME\&HTMLPAGE" length=linelen missover;
file   "&DIRNAME\Z&HTMLPAGE";

input @1 bigline $varying200. linelen;

   /*************************************/
   /* Read in and PUT new Selections.    */
   /*************************************/
if index(upcase(bigline),"<SELECT NAME=") and index(bigline,"&SELNAME") then do;

        put bigline;

        foundselect = 1;

        set &VARNAME nobs=totalobs;
         do i = 1 to totalobs;
             set &VARNAME point=i;
             put @1 string1;
                        readloop+1;
          end;

end;

if index(upcase(bigline),"<OPTION VALUE=") and foundselect = 1 then delete;
        else put @1 bigline;

if index(upcase(bigline),"</SELECT>") then foundselect = 0;

run;

*********************************************************;
* Build DOS commands to delete original file and rename  *;
* new file to old file name.                             *;
*********************************************************;
data _null_;

doublequote = '"';

deletefile = "del " || doublequote || "&DIRNAME\&HTMLPAGE" || doublequote;
call system(deletefile);

renamefile = "rename " || doublequote || "&DIRNAME\Z&HTMLPAGE" || doublequote
        || " " || doublequote || "&HTMLPAGE"|| doublequote;
call system(renamefile);

run;

    /*********************************************************************/
    /* End of GETFILES Macro.                                            */
    /*********************************************************************/
%MEND LOADHTML;

    /*********************************************************************/
    /* Execute the Macro to update selection list in HTML page.          */
    /*********************************************************************/
%LOADHTML(C:\TEMP,OrionStarReports.htm,REPORTYEAR,orsales,YEAR);

%LOADHTML(C:\TEMP,OrionStarReports.htm,PRODUCTGROUP,orsales,PRODUCT_GROUP);
```

Appendix B – The GETFILES SAS Macro

```
*************************************************************************;
* Program: GETFILES.sas                                                 *;
* This program reads a directory of .html report documents, and then    *;
* updates an HTML file with HREF links to those documents.              *;
*                                                                       *;
* This program is a SAS Macro with three parameters:                    *;
*      HTMLDIR - The directory of HTML files that are to be processed.  *;
*      HTMLFILE - The HTML file with the selection list that will be    *;
*                 updated by this program.                              *;
*      OUTPUTDIR - The directory that contains the HTMLFILE file that   *;
*                  will be updated.                                     *;
*************************************************************************;
%MACRO GETFILES(HTMLDIR, HTMLFILE, OUTPUTDIR);

options macrogen symbolgen mprint mlogic noxwait xsync;

    /*************************************************************************/
    /*                              Section I                                */
    /*************************************************************************/
    /* This section reads the specified HTML directory and builds HTML full-  */
    /* path addresses to the files stored there. They are saved in a SAS      */
    /* data set for use further on.                                           */
    /*************************************************************************/
filename htmldir "&HTMLDIR";

*************************************************************;
* Read the HTML directory, capture names of all .html files *;
* files, create data set of full-path names in html format. *;
*************************************************************;
data htmlnames;

length htmlname $40 htmlstring $200;
keep htmlstring;

retain optval '<A HREF="'
       endit  '">'
       boldon  '<B>'
       boldoff '</B>'
       break   '<BR>'
       enda    '</A>'
          slash   '\'
       ;

dirid = dopen("htmldir");

dircnt = dnum(dirid);

do i = 1 to dircnt;

        htmlname = dread(dirid,i);

        if index(htmlname,".html") then do;

                x = index(htmlname,".html");

                reptname = trim(left(substr(htmlname,1,x-1)));

                htmlstring = optval || "&HTMLDIR" || slash ||
                                       trim(left(htmlname)) ||
                                       endit || boldon ||
                                       trim(left(reptname)) ||
                                       boldoff || enda || break;

                output;

        end;

end;

run;
```

```
    /*************************************************************************/
    /*                              Section II                               */
    /*************************************************************************/
    /* This section reads the specified HTML web page and updates it with    */
    /* links to the web pages in the web directory specified by HTMLDIR.     */
    /*************************************************************************/

*****************************************;
* Process the old HTML file, creating new   *;
* one with updated HREFs.                   *;
*****************************************;
data _null_;

infile "&OUTPUTDIR.\&HTMLFILE" length=linelen missover;
file   "&OUTPUTDIR.\&HTMLFILE.z";

input @1 bigline $varying200. linelen;

if bigline =: '<A HREF=' then delete;
   else put   @1 bigline;

   /************************************/
   /* Read in file of updated HREFs.     */
   /************************************/
if bigline=:'<UL>' then do;

   set htmlnames nobs=totalcde;

   do i = 1 to totalcde;
       set htmlnames point=i;
       put @1 htmlstring;

   end;
end;

RUN;

*****************************************************************;
* Delete the original HTML file, rename the newly created HTML   *;
* file to the same name as the old one.                          *;
*****************************************************************;
data _null_;

call system("cd &OUTPUTDIR");  /* Allocate the HTML directory */
call system("del &HTMLFILE");  /* Delete old file from directory */
call system("ren &HTMLFILE.z &HTMLFILE"); /* Rename new file to old file name */

run;

%MEND GETFILES;
```

Appendix C – The Driver Program, and the IDENTIFY and SAS2HTM SAS Macros

```
*************************************************************************;
* Program: Driver_Program.sas                                           *;
*                                                                       *;
* This program %INCLUDEs several SAS Macro programs and then executes them  *;
* so that report files in specified directories can be read and then    *;
* rewritten to a specified directory as HTML files.                     *;
*************************************************************************;
*****************************************************************;
* SAS Options, Formats, etc.                                    *;
*****************************************************************;
options macrogen symbolgen mprint mlogic source source2;

*************************************************************************;
* INCLUDE the FILE2HTM.sas program that contains the FILE2HTM SAS Macro.*;
* The FILE2HTM macro turns a .txt report file into a .html file.        *;
*************************************************************************;
%include 'H:\Production SAS Programs\FILE2HTM.sas';

*************************************************************************;
* INCLUDE the SAS Macro program that identifies individual data files   *;
* in the specified report directory and builds macro calls that execute *;
* the FILE2HTM SAS Macro in the FILE2HTM.sas SAS program.               *;
*************************************************************************;
%include 'H:\ Production SAS Programs\IDENTIFY.sas';

*************************************************************************;
* Execute the IDENTIFY SAS Macro in the Identify.sas program to identify *;
* and process the data files in each specified data directory           *;
*************************************************************************;
%IDENTIFY(Z:\FlatFileReportDirectory,C:\Inetpub\ProductionReports);

*****************************************************************;
* NAME: IDENTIFY.sas                                            *;
* This macro reads a directory and creates an invocation of the  *;
* FILE2HTM SAS macro for each .txt file it finds in the directory*;
* The macro calls are stored in a temporary file until processing*;
* of the target directory is complete. Then the temporary file is*;
* %INCLUDEd so each of the macro calls are run to execute the    *;
* FILE2HTM macro for each .txt file.                            *;
*                                                               *;
* This macro accepts two parameters:                            *;
*  REPORTLIB - The directory with the .txt files                *;
*   HTMLOUT - The directory where the .html files will be written*;
*****************************************************************;

/*************************************************/
/* Beginning of the IDENTIFY SAS Macro.          */
/*************************************************/
%MACRO IDENTIFY(REPTLIB,HTMLLIB);

*****************************************************************;
* Pipe the names of report files to the SAS System for processing. *;
*****************************************************************;
filename dircmd pipe "dir /b ""&HTMLLIB\*.txt""";

*****************************************************************;
* Temporary file to hold the FILE2HTM SAS Macro call statements.   *;
*****************************************************************;
filename holdmacs TEMP;

*****************************************************************;
* Process each specific report file name, dropping unneeded ones.  *;
* Format SAS READFLAT Macro calls for valid data files.         *;
*****************************************************************;
data _null_;

length outline $300;

file holdmacs;

infile dircmd missover length=length;

input @;
```

```
input bigline $varying250. length;

outline = '%FILE2HTM(' || "&REPTLIB" || '\' || trim(left(bigline)) ||
',' || "&HTMLLIB" || '\' || tranwrd(trim(left(bigline)),".txt",".html")  || ');'
;

put outline;

run;

*****************************************************************;
* Include holdmacs, which is now a series of FILE2HTM SAS Macro    *;
* invocations. This will execute FILE2HTM.sas once for each ".txt" *;
*  flat file found in the target report directory.                 *;
*****************************************************************;
%INCLUDE holdmacs;

/************************************************/
/* End of the IDENTIFY SAS Macro.               */
/************************************************/
%MEND IDENTIFY;


*****************************************************************;
* NAME: FILE2HTM.sas                                            *;
* This macro accepts a .txt report file and writes it out to a  *;
* specified directory as a .html file.                          *;
*                                                               *;
* This macro accepts two parameters:                            *;
*    REPORTIN - The name of the input .txt report file.         *;
*     HTMLOUT - The name of the output .html file.              *;
*****************************************************************;
%MACRO FILE2HTM(REPORTIN,HTMLOUT);

************************************************;
* Allocate the input .txt file and the output .html*;
* file.                                          *;
************************************************;
FILENAME REPORTIN "&REPORTIN";
FILENAME HTMLOUT  "&HTMLOUT";

************************************************;
* Read the report .txt file. Write each record out *;
* to the .html file. Add HTML tags to the beginning*;
* and the end of the output file.                  *;
************************************************;
DATA _NULL_;

INFILE REPORTIN END=EOF truncover;

FILE HTMLOUT lrecl=84;

INPUT @1 STRING $char80.;

IF _N_ = 1 THEN DO;
   PUT "<HTML> <HEAD> </HEAD> <PRE> <BODY> <B>" ;
   PUT @1 STRING $char80. "</B>";
END;
ELSE    IF SUBSTR(STRING,1,1) = "0C"X THEN DO;
                      SUBSTR(STRING,1,1) = " ";
                      PUT '<P>';
               PUT '<CENTER><IMAGE SRC="images/hline.gif"></CENTER> <B>';
                      PUT @1 STRING $char80. "</B>";
               END;
               ELSE PUT @1 STRING $char80.;

IF EOF THEN DO;
   PUT "</BODY>";
   PUT "</PRE>";
   PUT "</HTML>";
END;

RUN;
```

```
  /*************************************************************/
  /* END OF FILE2HTM MACRO.                                    */
  /*************************************************************/
%MEND FILE2HTM;
```

Appendix D – The ODSHTML SAS Macro

```
*************************************************************************;
* Program: ODSHTML.sas                                                  *;
* This program uses ODS to create reports in HTML files and store them  *;
* in a specified directory on a web server.                             *;
*                                                                       *;
* This program is a SAS Macro with one parameter:                       *;
*   HTMLDIR - The directory where the HTML report files are to be stored.*;
*************************************************************************;

%MACRO ODSHTML(HTMLDIR);
/*********************************************************/
/* Allocate the SQL Server database.                     */
/*********************************************************/
libname sqllib oledb provider=sqlprov datasource=SQLPRD1
        user=joeuser properties=("Initial Catalog"=sastest)
       prompt=yes schema=dbo;

/*********************************************************/
/* Extract the data that will be used for the reports.   */
/*********************************************************/
proc sql;
       create table shoes as
       select * from sqllib.shoes
       order by saledate, region, subsidiary, product;
quit;

/*********************************************************/
/* Create the reports from the extracted data.           */
/*********************************************************/
Title1 "Quarterly Shoe Sale Reports";

ODS HTML BODY="&HTMLDIR\quarter_reports.html"  STYLE=Brick
         CONTENTS = "&HTMLDIR\quarter_contents.html"
         RAME = "&HTMLDIR\quarter_frame.html" ;

ODS PROCLABEL  "Shoe Sales for First Quarter 2007";

proc print data = shoes;
       by region subsidiary;
       id region subsidiary;
       where datepart(saledate) = '31MAR2007'D;
       var  product;
       sum stores sales;
title1 "Shoe Sales for First Quarter of 2007";
run;

ODS PROCLABEL  "Shoe Sales for Second Quarter 2007";

proc print data = shoes;
       by region subsidiary;
       id region subsidiary;
       where datepart(saledate) = '30JUN2007'D;
       var  product;
       sum stores sales;
title1 "Shoe Sales for Second Quarter of 2007";
run;

ODS PROCLABEL  "Shoe Sales for Third Quarter 2007";

proc print data = shoes;
       by region subsidiary;
       id region subsidiary;
       where datepart(saledate) = '30SEP2007'D;
       var  product;
       sum stores sales;
title1 "Shoe Sales for Third Quarter of 2007";
run;

ODS HTML CLOSE;
/*********************************************************/
/* End of the ODSHTML macro.                             */
/*********************************************************/
%MEND ODSHTML;
```

Appendix E – Program to Read SQL Server Tables and Create Descriptive Statistics in other SQL Server Tables

```
*************************************************************************;
* Program: Process and Load SQL Server Tables.sas                       *;
* This program reads SQL Server tables, performs elementary statistics  *;
* on specified variables, and then stores the statistics in other SQL   *;
* Server tables, so that it can be used for reports in web applications. *;
*************************************************************************;

/*********************************************************/
/* Program to load a SQL Server database with updated    */
/* values from other SQL Server tables.                  */
/*********************************************************/
libname sqllib oledb provider=sqlprov datasource=SQLPRD1
        user=joeuser properties=("Initial Catalog"=sastest)
       prompt=yes schema=dbo;

*****************************************************;
* Get unique values of YEAR from SQL Server database *;
*****************************************************;
proc sql;
       create table sqllib.Years_Totals as
       select year, sum(quantity) as Quantity_SUM,
                           min(quantity) as Quantity_MIN,
                           max(quantity) as Quantity_MAX,
                           mean(quantity) as Quantity_MEAN,
                           std(quantity) as Quantity_STD,
                           count(quantity) as Quantity_COUNT,
                           sum(profit) as Profit_SUM,
                           min(profit) as Profit_MIN,
                           max(profit) as Profit_MAX,
                           mean(profit) as Profit_MEAN,
                           std(profit) as Profit_STD,
                           count(profit) as Profit_COUNT,
                           sum(total_retail_price) as Retail_Price_SUM,
                           min(total_retail_price) as Retail_Price_MIN,
                           max(total_retail_price) as Retail_Price_MAX,
                           mean(total_retail_price) as Retail_Price_MEAN,
                           std(total_retail_price) as Retail_Price_STD,
                           count(total_retail_price) as Retail_Price_COUNT
       from sqllib.orsales
       group by year
       order by year;
quit;

*************************************************************;
* Get unique values of PRODUCT_CATEGORY from SQL Server database*;
*************************************************************;
proc sql;
       create table sqllib.Product_Group_Totals as
       select Product_Group,
                           min(quantity) as Quantity_MIN,
                           max(quantity) as Quantity_MAX,
                           mean(quantity) as Quantity_MEAN,
                           std(quantity) as Quantity_STD,
                           count(quantity) as Quantity_COUNT,
                           sum(profit) as Profit_SUM,
                           min(profit) as Profit_MIN,
                           max(profit) as Profit_MAX,
                           mean(profit) as Profit_MEAN,
                           std(profit) as Profit_STD,
                           count(profit) as Profit_COUNT,
                           sum(total_retail_price) as Retail_Price_SUM,
                           min(total_retail_price) as Retail_Price_MIN,
                           max(total_retail_price) as Retail_Price_MAX,
                           mean(total_retail_price) as Retail_Price_MEAN,
                           std(total_retail_price) as Retail_Price_STD,
                           count(total_retail_price) as Retail_Price_COUNT
       from sqllib.orsales
       group by Product_Group
       order by Product_Group;
quit;
```