

Make it SASsy: Using SAS® to Generate Personalized, Stylized, and Automated Email

Lisa Walter, Cardinal Health, Dublin, OH

Abstract

Email is everywhere! With the continuously growing number of outlets to stay connected with email, some could argue it is the best way to reach an audience. Knowing how to effectively integrate SAS with email messaging is beneficial to the end user, the business, and the programmer. This paper will explain how to use macro variables, the FILENAME statement, and some introductory HTML and CSS to create professional looking, personalized, automated emails. An experienced SAS user with a good understanding of the macro language will be able to integrate the discussed techniques into current reporting and messaging processes. The provided examples are designed using SAS 9.1.3 on a server environment. Screen shots of the resulting emails were viewed through Outlook 2003 with HTML enabled.

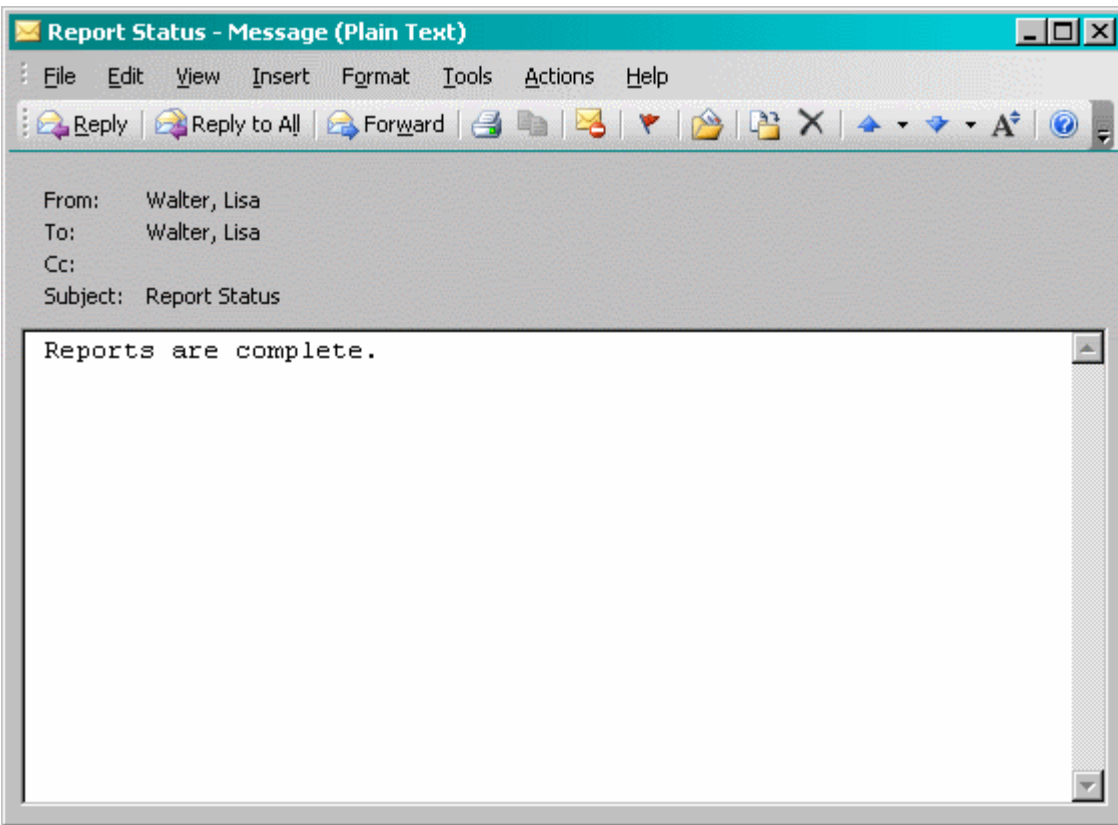
Introduction

Knowing how to generate personalized, stylized, and automated emails is simple if each task is addressed separately. In this paper, the process to create a sassy email will be broken into steps. A clear understanding of each step will help the reader apply the techniques to various circumstances. In addition, this introduction to smart emails provides the reader with a background that enables continued learning. First, the basic steps to generate an email using SAS will be reviewed. Then, the focus will shift to how to automate email language and distribution through the use of macro variables. Next, the reader will be introduced to HTML and CSS. The HTML and CSS will be integrated into the email to add the polished look. Finally, it will all be pulled together to create a truly SASsy email.

The Simple Email

SAS has built in the ability to send a simple email through the FILENAME statement. Below is a very basic example of using the FILENAME statement to send an email:

```
filename mymail email "LisaWalter@email.com"
  TYPE = "TEXT/PLAIN"
  SUBJECT = "Report Status";
data _null_;
  file mymail;
  put 'Reports are complete.';
run;
```



Let's take a look at each part of the syntax: First, the FILENAME statement is used to assign a new external file named 'mymail'. The option 'email' is set for the device-type of the external file. Another common device-type is 'printer'. The name enclosed in quotes designates the recipient of the email. This syntax is common practice; however, the statement could also be written TO = "LisaWalter@email.com". The TYPE option specifies the content-type of the email. This paper will examine other content-types a little later on. The SUBJECT option sets the subject of the email. Next, a _NULL_ DATA step is used. This is a special DATA step that is processed without creating a data set. The FILE statement defines the external file 'mymail' as the output destination for the DATA step. Within the DATA step, PUT statements are used to add content to the body of the email. Finally, terminate the DATA step using the RUN statement.

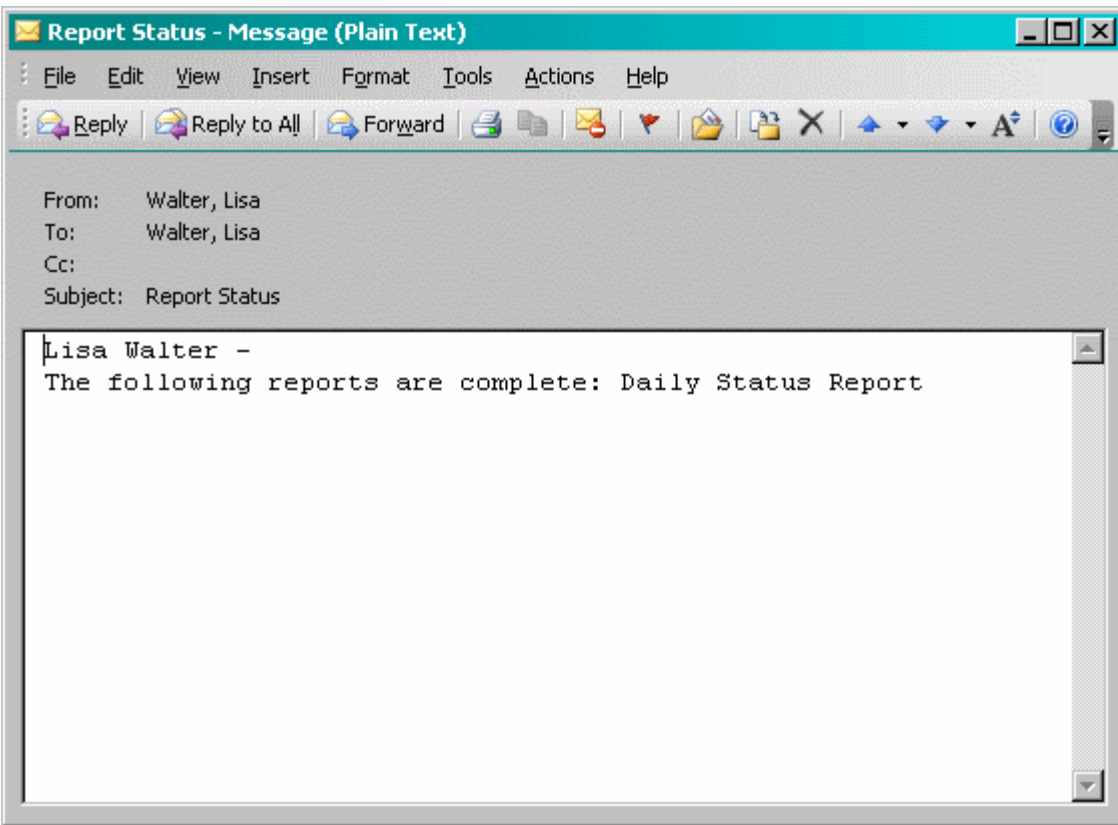
The FILENAME statement and the 'email' device-type contain a host of additional options. Some of these additional options will be introduced throughout the paper. Visit SAS support documentation for a full listing of options.

The Dynamic Email

The various options for the FILENAME statement as well as the body of the email, created through PUT statements, can be built using macro variables. This paper assumes a basic knowledge of macro variables. In most examples, macro variables will be set using a LET and SYMPUT statements; however, more practical applications will utilize data sets, DATA steps, and conditional logic to determine the value of the macro variables. Many wonderful papers demonstrating macro variable syntax are available through SAS.

```
%let to = LisaWalter@email.com;
%let name = Lisa Walter;
%let report = Daily Status Report;

filename mymail email "&to"
  TYPE = "TEXT/PLAIN"
  SUBJECT = "Report Status";
data _null_;
  file mymail;
  put "&name -";
  put "The following reports are complete: &report";
run;
```



With these few simple adjustments, this report is already more customized for the specific recipient. Again, the true leverage of utilizing macro variables in a meaningful way to customize email comes from utilizing data sets, DATA steps, and conditional logic to initialize the variables. The next example will demonstrate this technique. In addition, a new 'email' device-type option, CC, is introduced. CC sets a list of recipients to be carbon-copied on an email.

In this example, there are two data sets. The first data set, "reports", contains a list of all reports run. The second data set, "report_email_list", designates who receives each notification for each report. Using a few simple joins and a loop, we are able to send individualized emails for each person:

```

data reports;
  INPUT report $20.;
  DATALINES;
  Daily Status Report
  Usage Report
;
run;

data report_email_list;
  INPUT name $25.
         user_id $15.
         report $20.;
  DATALINES;
  Lisa Walter          LisaWalter    Daily Status Report
  Lisa Walter          LisaWalter    Usage Report
  Charlie Pierson     CharliePierson Daily Status Report
  Rachel Price        RachelPrice   Usage Report
;
run;

proc sql;
  SELECT count(DISTINCT user_id) into :numPeople
  FROM report_email_list;
  %let numPeople = &numPeople;
quit;

```

```

proc sql;
  SELECT DISTINCT name, user_id
    into :name1 - :name&numPeople, :userID1 - :userID&numPeople
  FROM report_email_list;
quit;

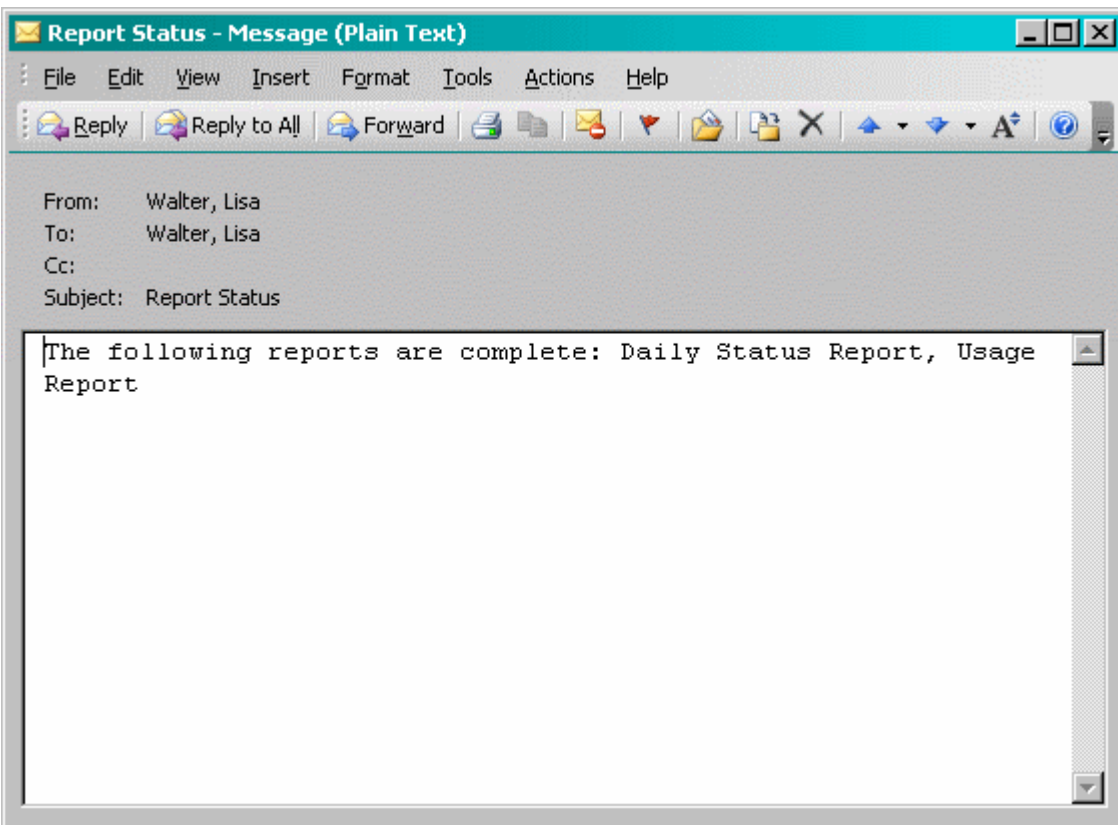
%macro send_report_emails();
  %do i=1 %to &numPeople;
    %let name = &&name&i;
    %let to = &&userID&i;

    *Creates a list of email addresses, quoted and separated by commas;
    proc sql;
      SELECT a.report into :report SEPARATED BY ', '
        FROM report_email_list as a
        JOIN reports as b
          ON a.report = b.report
        WHERE name = "&name"
          AND user_id = "&to";
    quit;

    filename mymail email "&to.@email.com"
      TYPE = "TEXT/PLAIN"
      SUBJECT = "Report Status";
    data _null_;
      file mymail;
      put "The following reports are complete: &report";
    run;
  %end;
%mend send_report_emails;

%send_report_emails;

```



Each individual was sent an email containing a list of completed reports. This code could also be altered to send each report in an individual email to the recipients specified on the report email table. This example is included in Appendix A.

The Stylized Email

HyperText Markup Language (HTML) and Cascading Style Sheets (CSS) are the primary languages used for web pages. HTML defines the elements displayed on a page and CSS determines the look and feel of the elements. These languages have been integrated into email to allow for more elaborate designs. Both languages are fairly simplistic and easy to learn. Many introductory websites and papers are available for free online. One example available online is W3Schools. For most email implementations, these free resources should be a sufficient learning tool. However, entire books have been written about both HTML and CSS. The brief introductions provided here, and those in many of the free resources, are not exhaustive of the entire languages.

All HTML tags, or in SAS terms, statements, have an opening and a closing syntax. Tags that create a visible item on the page are said to create an element. Generally, the format is written as <STATEMENT></STATEMENT>. Each tag could have a variety of opening and closing tags or text nested between. In addition, many tags have options, similar to SAS. One important option is class. The class will be used to designate style options using CSS. To get started, below is a list of common HTML tags that could be utilized in emails:

<html></html>	Indicates the start and end of HTML code. All other tags should be nested between these tags.
<head></head>	Provides an area for to provide additional, commonly unseen, information about the page. This is where we will include the CSS information.
<body></body>	All visible HTML elements are contained with these tags.
	Used to denote a subset of text or elements.
	Text nested between is bold.
<u></u>	Text nested between is underlined.
<i></i>	Text nested between is italicized.
 	Inserts a line break.
	Creates a bullet point list. This example is a single bullet point. Additional bullet points are created by using addition tags.
<table><tr><td></td></tr></table>	Creates a table. Each table row is created using the <tr></tr> tags and each table column is created using the <td></td> tags. All columns must be nested within a row. The example is a one row, one column table.

Where HTML tags are used to create elements on a web page, CSS is used to define the style of those elements. A style can be defined for a specific element, a group or class of elements, or a specific HTML tag. Additionally, CSS rules can be defined in a variety of locations for different scopes or levels. For simplicity, this paper will focus on document level CSS that only contains definitions for the classes and HTML tags. The basic syntax for the style definition is:

```
statement
{
  option: setting;
}
```

The statement in this instance can either be an HTML tag element or the name of a user defined class preceded by a dot, ".". The options define what to stylize and the setting defines how to style it. The following example demonstrates setting a style for a specific HTML tag's nested content:

```
body
{
  color: Purple;
  font-size: 10pt;
  font-family: Verdana;
}
```

The previous example will set all text nested within the HTML body tag to be of font family Verdana, size 10 point (pt), and colored purple: **Example**. As demonstrated above, the options for CSS are generally self-explanatory. With a

little research and after some use, determining the appropriate CSS options will come naturally. The next example demonstrates creating and using a class:

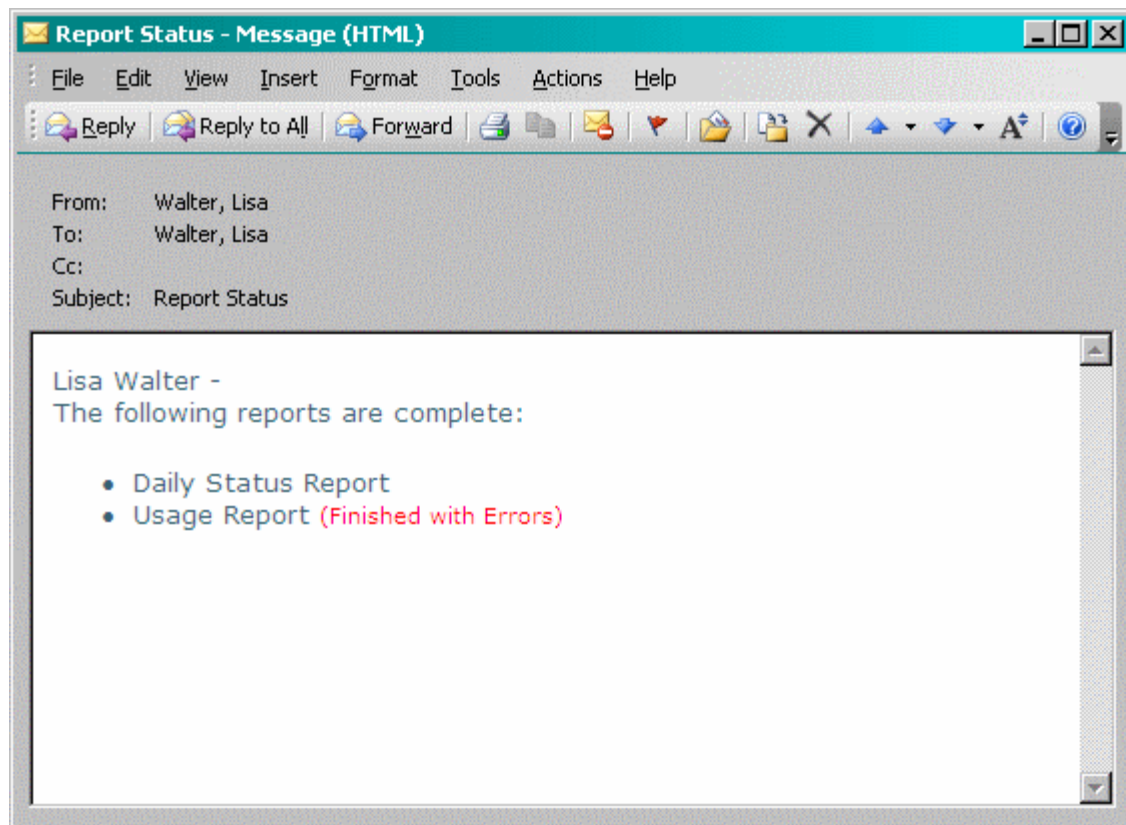
```
.errorMessage
{
  color: Red;
  font-size: 12pt;
  font-weight: bold;
}
```

```
<span class="errorMessage">Please contact your system administrator.</span>
```

Results in: **Please contact your system administrator.**

Integrating HTML and CSS into a SAS generated email is simple. The only necessary step is to change the TYPE option from 'Plain/Text' to 'Plain/HTML'! The rest of the work comes in the body of the email using the PUT statements. To integrate HTML and CSS in emails the <html> and <body> tags must be included in the body of the email. CSS information should be added with the <head> tag with a nested <style> tag. The following example adds basic styling to an email:

```
filename mymail email "LisaWalter@email.com"
  TYPE='TEXT/HTML'
  SUBJECT = "Report Status";
data _null_;
  file mymail;
  put '<html><head>';
  put '<style type="text/css" MEDIA=screen><!--';
  put 'body { color: #346170; font-family: Verdana; font-size: 10pt; }';
  put '.errorMessage { color: Red; font-size: 8pt; }';
  put '--></style></head><body>';
  put 'Lisa Walter -<br /> The following reports are complete:';
  put '<ul><li>Daily Status Report</li>';
  put '<li>Usage Report <span class="errorMessage">(Finished with';
  put 'Errors)</span></li></ul>';
  put '</body></html>';
run;
```



The above example only utilizes the two CSS examples and some of the HTML tags explained above. Making a large impact does not need to take a lot of additional code!

The Sassy Email

The techniques demonstrated above can be combined to create stylized email with dynamically generated distribution. The following example uses the same data sets referenced in the macro variable section, except the reports table now has an additional column indicating whether the report run was successful. If the report run was not successful, the value "Error" appears in the new status column.

```

proc sql;
  SELECT count(DISTINCT user_id) into :numPeople
  FROM report_email_list;
  %let numPeople = &numPeople;
quit;

proc sql;
  SELECT DISTINCT name, user_id
  into :name1 - :name&numPeople, :userID1 - :userID&numPeople
  FROM report_email_list;
quit;

%macro send_report_emails();
  %do i=1 %to &numPeople;
    %let name = &&name&i;
    %let to = &&userID&i;

    *Creates a list of email addresses, quoted and seperated by commas;
    proc sql;
      SELECT count(*) into :numReports
      FROM report_email_list as a
      JOIN reports as b
      ON a.report = b.report
      WHERE name = "&name"
    
```

```

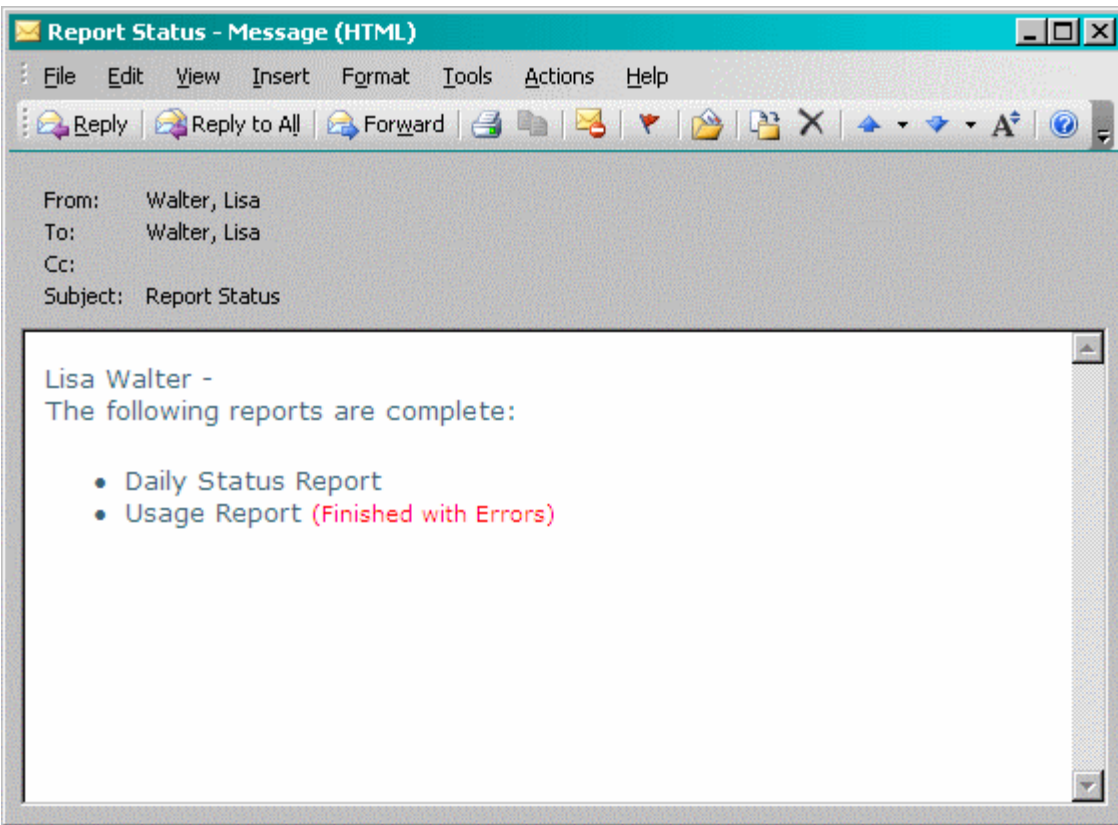
        AND user_id = "&to";
        %let numReports = &numReports;
quit;

proc sql;
    SELECT a.report, status
        into :report1 - :report&numReports, :status1 - :status&numReports
    FROM report_email_list as a
    JOIN reports as b
        ON a.report = b.report
    WHERE name = "&name"
        AND user_id = "&to";
quit;

filename mymail email "&to.@email.com"
    TYPE = "TEXT/HTML"
    SUBJECT = "Report Status";
data _null_;
    file mymail;
    put '<html><head>';
    put '<style type="text/css" MEDIA=screen><!--';
    put 'body { color: #346170; font-family: Verdana; font-size: 10pt; }';
    put '.errorMessage { color: Red; font-size: 8pt; }';
    put '--></style></head><body>';
    put 'Lisa Walter -<br /> The following reports are complete:';
    put '<ul>';
    %do j=1 %to &numReports;
        put "<li>&&report&j";
        %if "&&status&j" = "Error" %then %do;
            put '<span class="errorMessage"> (Finished with Errors)</span>';
        %end;
        put '</li>';
    %end;
    put '</ul>';
    put '</body></html>';
run;
%end;
%mend send_report_emails;

%send_report_emails;

```

Conclusion

Throughout this paper, simple examples were provided to demonstrate how combining SAS, HTML, and CSS can create professional looking emails. Automated reports allow team members to stay connected with the status of processes already implemented in the business. In addition, the appearance of these automated emails leaves a strong impression on the reader about the quality of data and the professionalism of the sender. Next time, before sending that email, take the few extra steps to ensure emails are not only accurate, but also sassy.

References

- Bahler, Caroline, and Eric Brinsfield. "Report Creation Using Data _NULL_." *SUGI 27 Paper 61-27*. SAS Inc. Web. <<http://www2.sas.com/proceedings/sugi27/p061-27.pdf>>.
- "SAS Functions and CALL Routines under Windows FILENAME Function: Windows." *SAS(R) 9.2 Companion for Windows, Second Edition*. SAS Inc. Web. <<http://support.sas.com/documentation/cdl/en/hostwin/63285/HTML/default/viewer.htm#/documentation/cdl/en/hostwin/63285/HTML/default/win-func-filename.htm>>.
- "SAS Statements under Windows FILENAME Statement: Windows." *SAS(R) 9.2 Companion for Windows, Second Edition*. SAS Inc. Web. <<http://support.sas.com/documentation/cdl/en/hostwin/63285/HTML/default/viewer.htm#/documentation/cdl/en/hostwin/63285/HTML/default/chfnoptfmain.htm>>.
- "Sending E-Mail from Within SAS." Web. <<http://www.uc.edu/sashtml/cms/z1271256.htm>>.
- Worden, Jeanina, and Philip Jones. "You've Got Mail – E-mailing Messages and Output Using SAS® EMAIL Engine." *SUGI 29 Paper 178-29*. SAS Inc. Web. <<http://www2.sas.com/proceedings/sugi29/178-29.pdf>>.

Contact Information

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

Lisa Walter
Cardinal Health
7000 Cardinal Pl.
Dublin, OH 43017
Phone: 614.553.4978

E-mail: lisa.walter01@gmail.com

Appendix A

The below example creates two data sets. The first data set, "reports", contains a list of all reports run. The second data set, "report_email_list", designates who receives each notification for each report. Using a few simple joins and a loop, we are able to send individualized emails for each report:

```
data reports;
  INPUT report $20.;
  DATALINES;
  Daily Status Report
  Usage Report
;
run;

data report_email_list;
  INPUT name $25.
        user_id $15.
        report $20.;
  DATALINES;
  Lisa Walter      LisaWalter      Daily Status Report
  Lisa Walter      LisaWalter      Usage Report
  Charlie Pierson  CharliePierson Daily Status Report
  Rachel Price     RachelPrice     Usage Report
;
run;

proc sql;
  SELECT count(*) into :numReports
  FROM reports;
  %let numReports = &numReports;
quit;

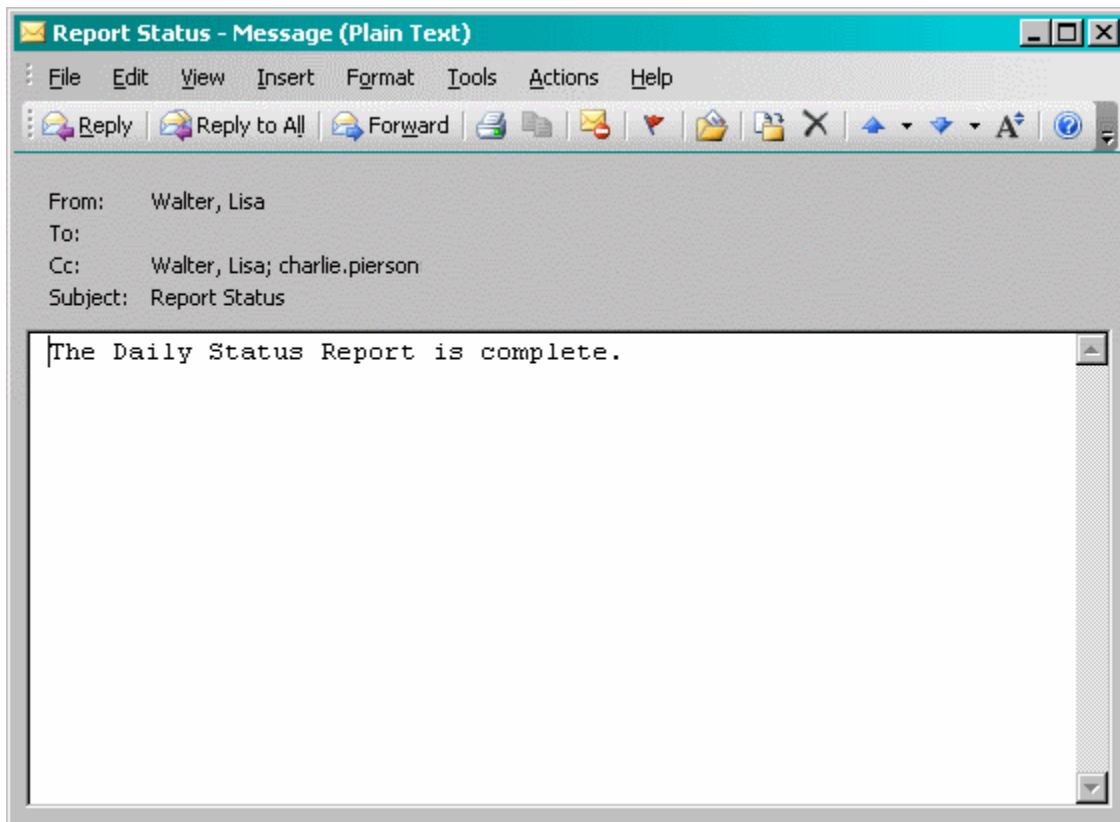
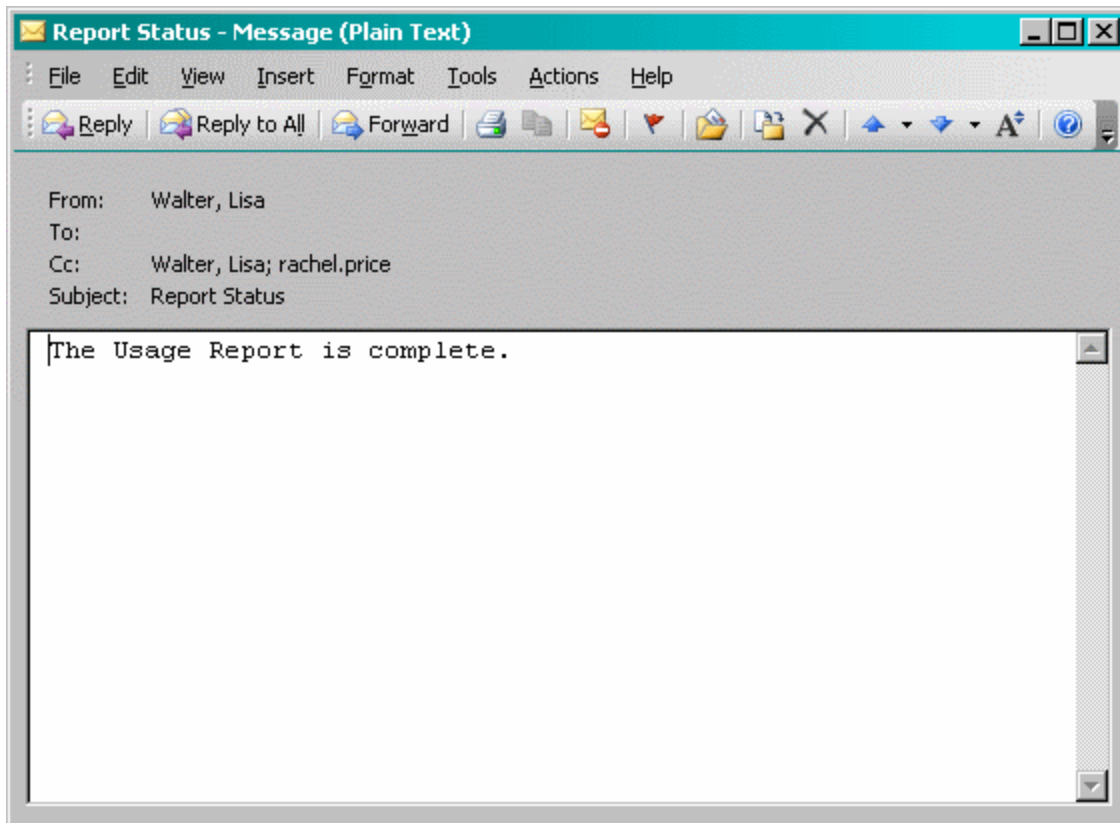
proc sql;
  SELECT report into :report1 - :report&numReports
  FROM reports;
quit;

%macro send_report_emails();
  %do i=1 %to &numReports;
    %let report = &&report&i;

    *Creates a list of email addresses, quoted and separated by commas;
    proc sql;
      SELECT compress(user_id) || '@email.com' into :to SEPARATED BY ' ' , ''
      FROM report_email_list
      WHERE report = "&report";
    quit;

    filename mymail email
      TYPE = "TEXT/PLAIN"
      SUBJECT = "Report Status"
      CC = ( "&to" );
    data _null_;
      file mymail;
      put "The &report is complete.";
    run;
  %end;
%mend send_report_emails;

%send_report_emails;
```



Appendix B

Below are a two other examples of emails achieved using the techniques demonstrated above:

New Approval Request: Sample Email - Message (HTML)

File Edit View Insert Format Tools Actions Help


Reply Reply to All Forward [Print] [Folder] [Envelope] [Heart] [Folder] [X] [Up] [Down] [A+] [Help]

From:

To: fake.guy

Cc:

Subject: New Approval Request: Sample Email

Attachments:  Sample_Email_pva.rtf (230 KB)

Ashley Kline-Tozzi has submitted an approval request for Sample Email (NEW000000188). The standard PVA is attached. Approval is due 08/19/10.

Ashley Kline-Tozzi 's Justification:
N/A

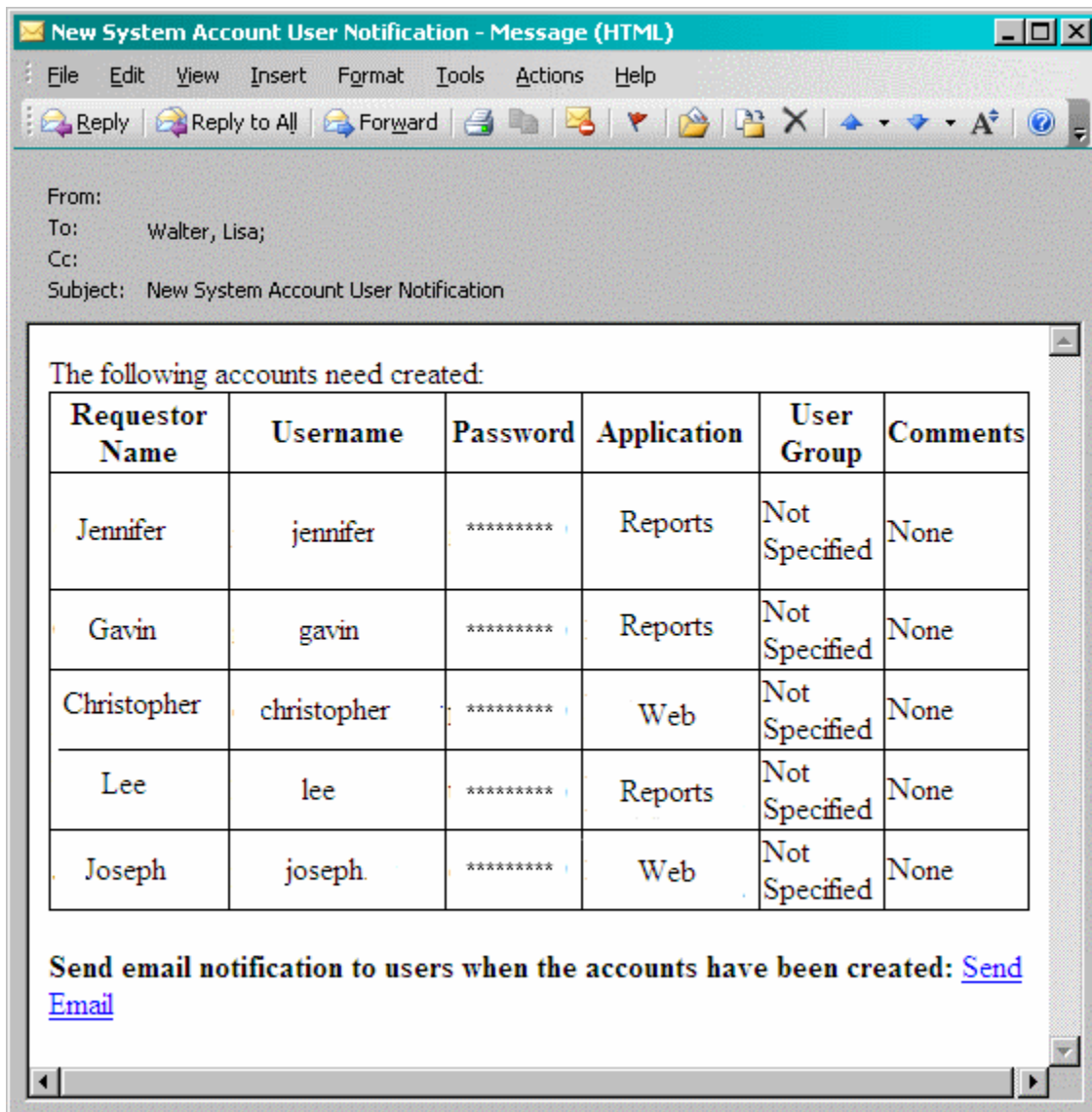
Key Metrics:

- Annual Net Sales: \$6,600.00
- Gx Source Volume: \$79,200.00
- Upfront Discount: \$0.00 (0.00 %)
 - Year 1: \$0.00 (0.00 %)

The approvers are as follows:

Approver	Title	Function	Response	Comments
Fake Guy	Manager	Sales	Pending	N/A

Rejection Comments:



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.