Paper 9-27

ODS Meets SAS/IntrNet®

Susan J. Slaughter, Avocet Solutions, Davis, CA

Sy Truong, Meta-Xceed, Inc, Fremont, CA

Lora D. Delwiche, University of California, Davis

Abstract

The SAS® System gives you the ability to create a wide range of web-ready reports. This paper walks through a series of examples showing what you can do with Base SAS and when you need SAS/IntrNet. Starting with the simplest HTML reports, this paper shows how you can jazz up your output by using the STYLE= options, traffic-lighting and hyper-linking available in the reporting procedures: PRINT, REPORT, and TABULATE. With the use of SAS/IntrNet you can add functionality to reports with features such as drill-down links that are data-driven, and you can produce dynamic reports created on-the-fly for individual users. Using this technique, your clients can navigate to the exact information needed to fulfill your business objective.

Introduction

In the SAS System, both the Output Delivery System (ODS) and SAS/IntrNet produce documents for viewing over the Internet. All SAS users have ODS because it is part of Base SAS, but SAS/IntrNet is a separate product which you must have installed in addition to Base SAS.

If all you want to do is produce reports and post them on the Internet for people to view, then you probably don't need SAS/IntrNet. With a few ODS statements you can send any SAS output to the HTML (Hyper Text Markup Language) destination. You can also change the way HTML output looks by choosing one of the built-in style definitions that comes with Base SAS, or creating your own style definition using the TEMPLATE procedure. (Unfortunately, we don't have room to cover PROC TEMPLATE in this paper. For more information on PROC TEMPLATE or ODS basic concepts, see Slaughter and Delwiche (2001).) Using the STYLE= option in the TABULATE, REPORT, and PRINT procedures, you can change the color, font, and many other features of reports. You can even insert images and hyperlinks.

Using ODS to insert hyperlinks, you can create a pseudo-dynamic effect. When a person clicks on one of these hyperlinks, the browser takes them to a new page. While this has a dynamic feel, the new page is in fact static because you have created it in advance. To create truly dynamic reports, you need SAS/IntrNet.

With SAS/IntrNet you can create reports on the fly, based on the needs of individual users. The advantage of combining SAS/IntrNet with your ODS programs is that your program is dynamically executed when the user clicks on your hyperlink. This means that if your data is changing, the drill-down will capture the most up-to-date results. If your reports are not time dependent, the static approaches of generating HTML reports with ODS will suffice. However, if your report helps decision makers decide upon time-sensitive information, the marriage of ODS and SAS/IntrNet is the perfect solution.

The Data

The data used in this paper come from Jelly World, a fictitious manufacturer of gourmet jelly beans. Jelly World has three factories where they are producing five new flavors. For each factory and day, the SAS data set contains the flavor produced, and the pounds (in millions). Selected observations from the SAS data set appear in Table 1.

Table 1. Selected observations from the SAS data set Production.

```
                Jelly Bean Production Data

 Obs     Flavor   Factory       Date      MPounds

 350       M         A      12/17/01     0.039755
 351       M         A      12/18/01     0.042564
 352       P         A      12/19/01     0.046849
 357       P         B      01/02/01     0.036736
 361       P         B      01/06/01     0.039049
 362       C         B      01/07/01     0.036954
 363       M         B      01/08/01     0.042112
```

The Basic Table

For the first part of this paper we will be using basically the same table produced from PROC TABULATE to show you how you can use ODS and the various STYLE options to modify the look of the table. Here are the SAS statements that produce this basic table and the listing output is shown in Table 2.

```
PROC FORMAT;
  VALUE $flav
       'P' = 'Pecan Pie'
        'B' = 'Banana Bash'
        'A' = 'Apple Spice'
        'M' = 'Mango'
        'C' = 'Choco Mint';
TITLE 'Jelly Bean Production in 2001';
TITLE2 'Millions of Pounds';
PROC TABULATE DATA=production FORMAT=4.1;
CLASS Factory Flavor;
   VAR MPounds;
   FORMAT Flavor $flav.;
   TABLE Flavor ALL,
         Factory*SUM=''*MPounds=''
             ALL*SUM=''*MPounds='';
RUN;
```

Table 2. Listing output from basic PROC TABULATE.

```
                Jelly Bean Production in 2001
                     Millions of Pounds

 ------------------------------------------------
 |                       |   Factory   |      |
 |                       |-------------|      |
 |                       | A  | B  | C  | All  |
 |-----------------------+----+----+----+------|
 |Flavor                 |    |    |    |      |
 |-----------------------|    |    |    |      |
 |Apple Spice            | 2.8| 2.8| 4.6|10.2  |
 |-----------------------+----+----+----+------|
 |Banana Bash            | 3.9| 3.2| 2.1|  9.1 |
 |-----------------------+----+----+----+------|
 |Choco Mint             | 0.9| 5.0| 1.7|  7.5 |
 |-----------------------+----+----+----+------|
 |Mango                  | 3.3| 2.3| 4.3|  9.9 |
 |-----------------------+----+----+----+------|
 |Pecan Pie              | 4.3| 1.9| 2.5|  8.8 |
 |-----------------------+----+----+----+------|
 |All                    |15.1|15.3|15.1|45.5  |
 ------------------------------------------------
```

Using ODS Statements to Create HTML Output

To generate HTML output all you need are two statements—one to open the HTML file, and one to close it. There are other types of ODS statements, and other types of HTML files that SAS can write, not to mention other types of output. However, in this example we simply want to create an HTML file containing the report. You do that with these two basic statements:

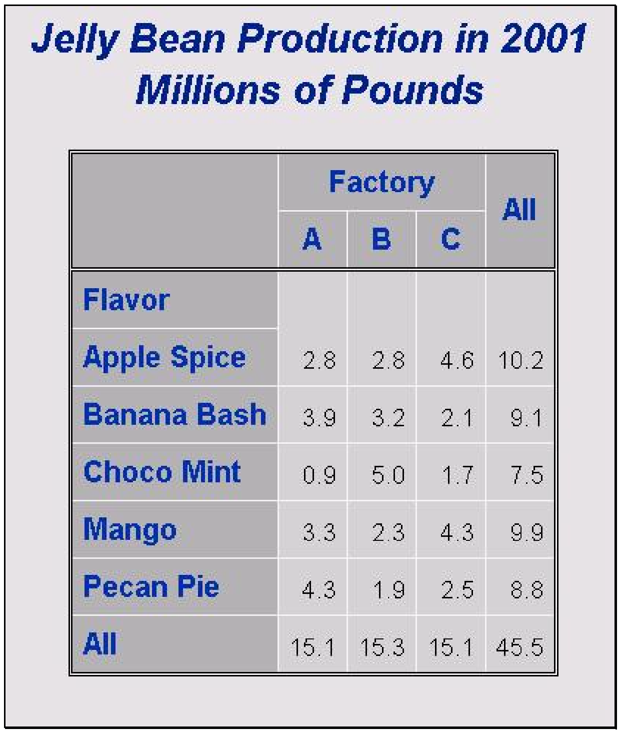
```
ODS HTML BODY = 'bodyfile.html'
   STYLE = style-name;

ODS HTML CLOSE;
```

The ODS HTML statement opens a file, specified in the BODY= option, which will contain the report. You can also use the FILE= option as an alias for BODY=. The ODS HTML CLOSE statement closes the file when the report is done. Generally speaking, you want to put the first statement just before the procedure and the closing statement just after the RUN statement. Here is the TABULATE procedure used earlier, with ODS statements added.

```
ODS HTML BODY = 'c:\sugi\jellybean2.html';

PROC TABULATE DATA=production FORMAT=4.1;
   CLASS Factory Flavor;
   VAR MPounds;
   FORMAT Flavor $flav.;
   TABLE Flavor ALL,
         Factory*SUM=''*MPounds=''
             ALL*SUM=''*MPounds='';
RUN;

ODS HTML CLOSE;
```

When you run this program, you get the output in Table 3.

Table 3. HTML output from a simple PROC TABULATE using the DEFAULT style definition.



Jelly Bean Production in 2001
Millions of Pounds

	Factory			All
	A	B	C	
Flavor				
Apple Spice	2.8	2.8	4.6	10.2
Banana Bash	3.9	3.2	2.1	9.1
Choco Mint	0.9	5.0	1.7	7.5
Mango	3.3	2.3	4.3	9.9
Pecan Pie	4.3	1.9	2.5	8.8
All	15.1	15.3	15.1	45.5

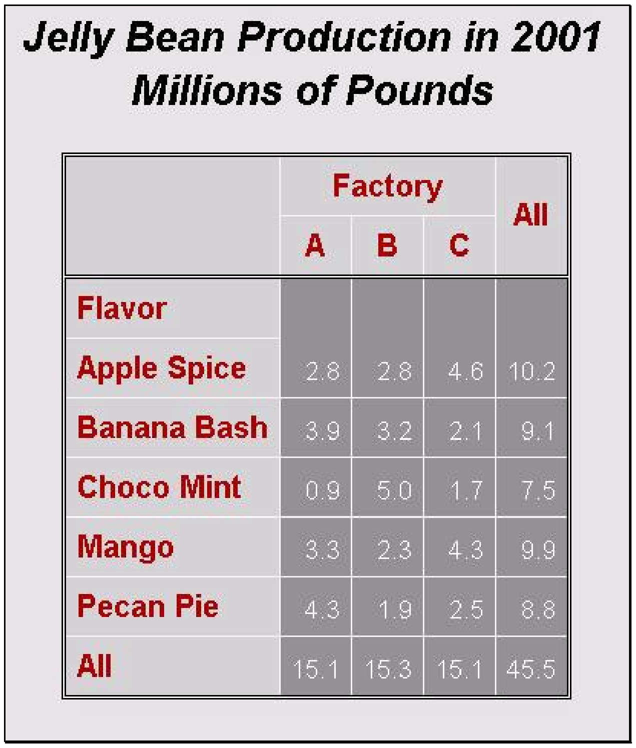
Changing Style Definitions

If you don't like the DEFAULT style, you can choose another one by simply adding the STYLE= option to the opening ODS statement. To use the BRICK style definition, for example, you could substitute this statement in the preceding program:

```
ODS HTML BODY = 'c:\sugi\jellybean2.html'
   STYLE = BRICK;
```

Table 4 shows the output using the BRICK style definition. To see a list of all the styles available to you, select **Tools-Options-Preferences** from the menus and click on the **Results** tab. Then look in the drop-down list for Style.

Table 4. HTML output from PROC TABULATE using the BRICK style definition.



Jelly Bean Production in 2001
Millions of Pounds

	Factory			All
	A	B	C	
Flavor				
Apple Spice	2.8	2.8	4.6	10.2
Banana Bash	3.9	3.2	2.1	9.1
Choco Mint	0.9	5.0	1.7	7.5
Mango	3.3	2.3	4.3	9.9
Pecan Pie	4.3	1.9	2.5	8.8
All	15.1	15.3	15.1	45.5

Adding Style to TABULATE Output

We have seen how you can use the STYLE= option on the ODS statement to change the overall look of your output. But, with the TABULATE procedure, you can also use the STYLE= option on various TABULATE statements to change the style of specific parts of your table. You can also use STYLE= options in PROC REPORT and, with release 8.2 of the SAS system, you can use the STYLE= option in PROC PRINT as well. Table 5 shows which PROC TABULATE statements can use the STYLE= option and which parts of the table are affected.

Table 5. TABULATE statements that can use the STYLE= option.

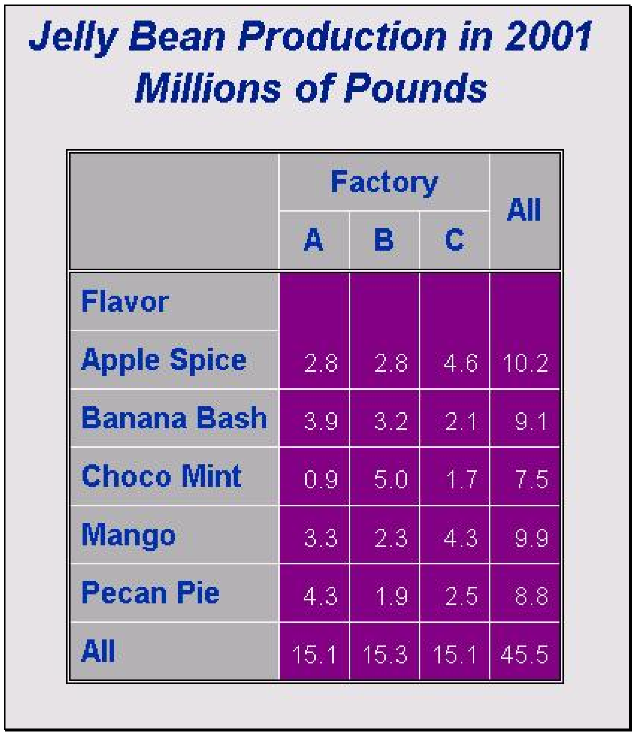
Statement	Table region affected
PROC TABULATE	Data cells
CLASS	Class variable name headings
CLASSLEV	Class level value headings
KEYWORD	Keyword headings
TABLE (as an option)	Table borders
TABLE (crossed with elements)	Element's data cell
VAR	Analysis variable name headings

The following program adds the STYLE= option to the PROC TABULATE statement while keeping the other statements the same as in the last example. Here we are changing the background color of the data cells to purple and the foreground (the text) to white.

```
ODS HTML BODY='c:\sugi\jellybean3.html';

PROC TABULATE DATA=production FORMAT=4.1
     STYLE={BACKGROUND=purple FOREGROUND=white};
    CLASS Factory Flavor;
    VAR MPounds;
    FORMAT Flavor $flav.;
    TABLE Flavor ALL,
          Factory*SUM=''*MPounds=''
           ALL*SUM=''*MPounds='';
RUN;

ODS HTML CLOSE;
```

The HTML output from this program appears in Table 6. This output uses the DEFAULT style. Notice how all the data cells are affected by the change.

Table 6. TABULATE HTML output using the DEFAULT style and the STYLE= option on the PROC TABULATE statement.



Jelly Bean Production in 2001
Millions of Pounds

	Factory			All
	A	B	C	
Flavor				
Apple Spice	2.8	2.8	4.6	10.2
Banana Bash	3.9	3.2	2.1	9.1
Choco Mint	0.9	5.0	1.7	7.5
Mango	3.3	2.3	4.3	9.9
Pecan Pie	4.3	1.9	2.5	8.8
All	15.1	15.3	15.1	45.5

There are many different attributes you can specify. Table 7 lists some of the attributes you can change and examples of valid values for those attributes. For a complete list, see the *SAS Procedures Guide, Version 8*, Chapter 2, "Fundamental Concepts for Using Base SAS Procedures".

Table 7. Selected attributes and examples of valid values.

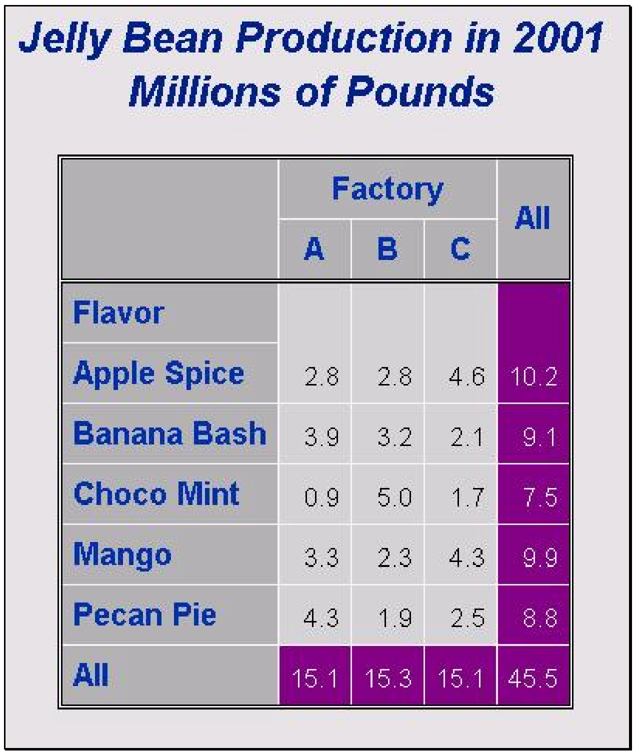
Attribute	Example of Valid Value
BACKGROUND	Black
BACKGROUNDIMAGE	'c:\images\mybackground.jpg'
BORDERCOLOR	Yellow
FLYOVER	'Good Work!'
FONT	Courier
FOREGROUND	White
JUST	Center
PREIMAGE	'c:\images\logo.gif'
URL	'c:\documents\salesreps.html'

The following program, instead of using the STYLE= option on the PROC TABULATE statement, crosses the STYLE= option with an element of the TABLE statement. The ALL keywords in the row and column dimensions are crossed with the style.

```
ODS HTML BODY='c:\sugi\jellybean4.html';

PROC TABULATE DATA=production FORMAT=4.1;
   CLASS Factory Flavor;
   VAR MPounds;
   FORMAT Flavor $flav.;
   TABLE Flavor ALL*{STYLE=
      {BACKGOUND=purple FOREGROUND=white}},
      Factory*SUM=''*MPounds=''
      ALL*SUM=''*MPounds=''*{STYLE=
      {BACKGROUND=purple FOREGROUND=white}};
RUN;

ODS HTML CLOSE;
```

The HTML output from this program appears in Table 8. Notice how now only the data cells corresponding to the ALL row and column have the new style. By placing the STYLE= option in the row or column dimension of the TABLE statement, you have more control over which data cells are affected by the style.

Table 8. TABULATE HTML output using the DEFAULT style and the STYLE= option crossed with the ALL keywords in the TABLE statement.



Jelly Bean Production in 2001
Millions of Pounds

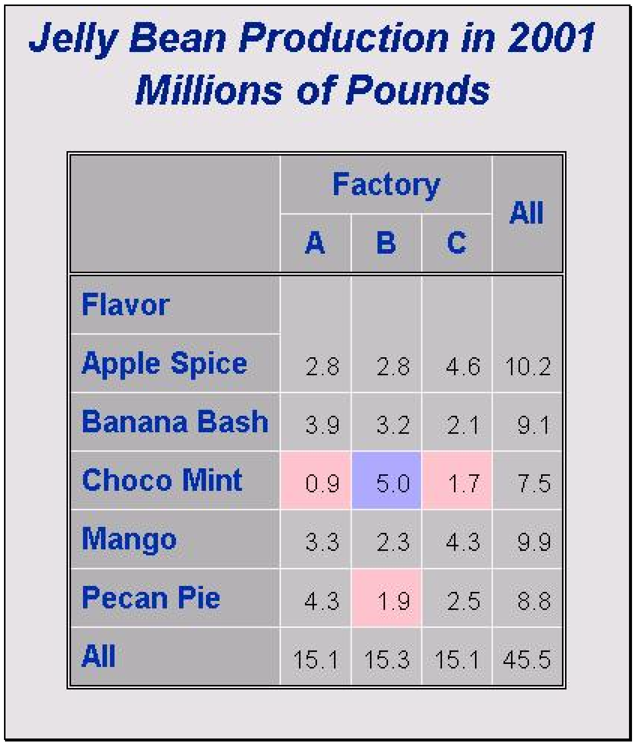
	Factory			All
	A	B	C	
Flavor				
Apple Spice	2.8	2.8	4.6	10.2
Banana Bash	3.9	3.2	2.1	9.1
Choco Mint	0.9	5.0	1.7	7.5
Mango	3.3	2.3	4.3	9.9
Pecan Pie	4.3	1.9	2.5	8.8
All	15.1	15.3	15.1	45.5

You can have even more control over the style of the data cells by using a feature known as traffic-lighting. Here the value of the data cell determines the style that the data cell displays. Using traffic-lighting, you can draw attention to important values, or highlight relationships between values. To add traffic-lighting, create a user-defined format specifying different style attributes for each range of values, and then use that format as the value of the style element.

The following program defines a format, PROD. which assigns colors to value ranges. We want to highlight the table cells where production of the flavor is too low (<2) and where production is too high (between 5 and 7). We also set the background color of the cells over 7 to light gray because we don't want the ALL row and column colored. To apply this format, set the value of the BACKGROUND= style element equal to the PROD. format. This style is applied to the whole table by placing the STYLE= option in the PROC TABULATE statement. The results of this program are shown in Table 9.

```
ODS HTML BODY='c:\sugi\jellybean5.html';
PROC FORMAT;
   VALUE prod
        0-<2 = 'pink'
       2-<5 = 'ltgray'
       5-<7 = 'very light blue'
       7-HIGH='ltgray';
PROC TABULATE DATA=production FORMAT=4.1
      STYLE={BACKGROUND=prod.};
   CLASS Factory Flavor;
   VAR MPounds;
   FORMAT Flavor $flav.;
   TABLE Flavor ALL,
         Factory*SUM=''*MPounds=''
         ALL*SUM=''*MPounds='';
RUN;
ODS HTML CLOSE;
```

Table 9. TABULATE HTML output using the DEFAULT style and traffic-lighting.



Jelly Bean Production in 2001
Millions of Pounds

	Factory			All
	A	B	C	
Flavor				
Apple Spice	2.8	2.8	4.6	10.2
Banana Bash	3.9	3.2	2.1	9.1
Choco Mint	0.9	5.0	1.7	7.5
Mango	3.3	2.3	4.3	9.9
Pecan Pie	4.3	1.9	2.5	8.8
All	15.1	15.3	15.1	45.5

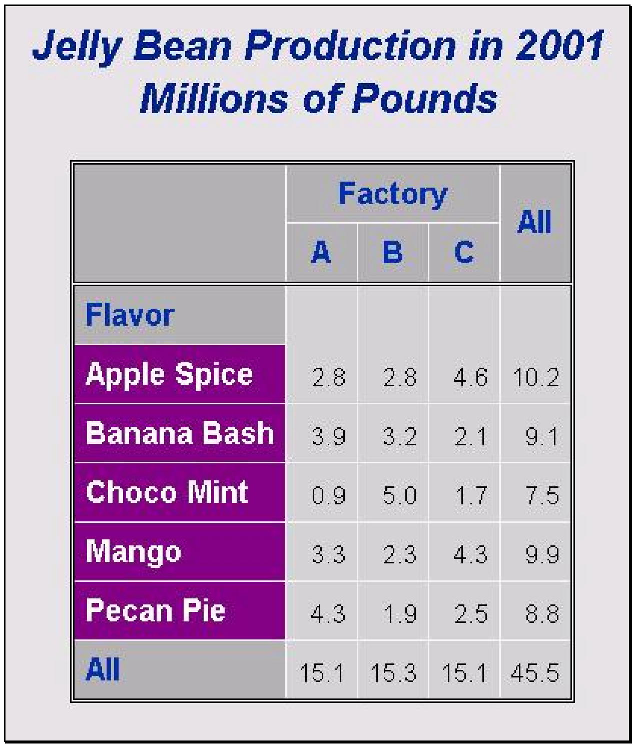
So far we have only changed the style of the data cells by placing the STYLE= option either in the PROC TABULATE statement or by placing it in a TABLE statement crossed with another element. You can also change the style of other regions of the table by placing the STYLE= option in other statements (see Table 5).

The CLASSLEV statement allows you to specify styles for the headings of the levels of the class variables. The following program uses the STYLE= option on the CLASSLEV statement to change the style of the headings for the class variable Flavor.

```
ODS HTML BODY='c:\sugi\jellybean6.html';
PROC TABULATE DATA=production FORMAT=4.1 ;
   CLASS Factory Flavor;
   CLASSLEV Flavor / STYLE={BACKGROUND=purple
FOREGROUND=white};
   VAR MPounds;
   FORMAT Flavor $flav.;
   TABLE Flavor ALL,
         Factory*SUM=''*MPounds=''
         ALL*SUM=''*MPounds='';
RUN;
ODS HTML CLOSE;
```

The results of this program are shown in Table 10. Notice that none of the data cells are affected by the CLASSLEV statement, only the headings.

Table 10. TABULATE HTML output using the DEFAULT style and the STYLE= option on the CLASSLEV statement.



Jelly Bean Production in 2001
Millions of Pounds

	Factory			All
	A	B	C	
Flavor				
Apple Spice	2.8	2.8	4.6	10.2
Banana Bash	3.9	3.2	2.1	9.1
Choco Mint	0.9	5.0	1.7	7.5
Mango	3.3	2.3	4.3	9.9
Pecan Pie	4.3	1.9	2.5	8.8
All	15.1	15.3	15.1	45.5

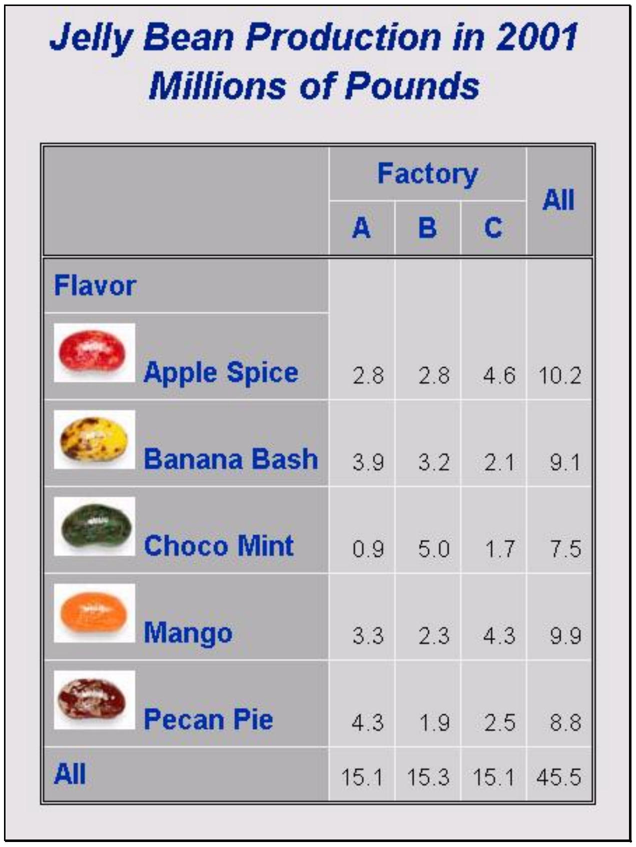
One of the interesting things you can do with the CLASSLEV statement is to include images in the headings for the class variables. To do this, create a format that assigns the file names of your images to the different levels of your class variable. Then set the PREIMAGE style attribute equal to this format in the STYLE= option. Starting with version 8.2, the PREIMAGE attribute is valid for all destinations except listing and output.

The following program creates a format, $IMAGE., that assigns file names to the different values of the Flavor variable. Then the PREIMAGE style element is set equal to the $IMAGE. format in the CLASSLEV statement.

```
ODS HTML BODY='c:\sugi\jellybean7.html';
PROC FORMAT;
  VALUE $image
       'P' = 'C:\images\pecanpie.jpg'
       'B' = 'C:\images\bananabash.jpg '
       'A' = 'C:\images\applespice.jpg'
       'M' = 'C:\images\mango.jpg '
       'C' = 'C:\images\chocomint.jpg';
PROC TABULATE DATA=production FORMAT=4.1;
   CLASS Factory Flavor;
   CLASSLEV Flavor/STYLE={PREIMAGE=$image.};
   VAR MPounds;
   FORMAT Flavor $flav.;
   TABLE Flavor ALL,
         Factory*SUM=''*MPounds=''
           ALL*SUM=''*MPounds='';
RUN;
ODS HTML CLOSE;
```

The HTML output resulting from this program is shown in Table 11. You can see the pictures of the various jelly beans are now included in the headings of the table. Obviously you would want to keep your images pretty small if you are going to include them in a table. Jelly beans, it turns out, fit quite nicely in a table.

Table 11. TABULATE HTML output using the DEFAULT style and the PREIMAGE attribute of the STYLE= option in the CLASSLEV statement.



Jelly Bean Production in 2001
Millions of Pounds

	Factory			All
	A	B	C	
Flavor				
Apple Spice	2.8	2.8	4.6	10.2
Banana Bash	3.9	3.2	2.1	9.1
Choco Mint	0.9	5.0	1.7	7.5
Mango	3.3	2.3	4.3	9.9
Pecan Pie	4.3	1.9	2.5	8.8
All	15.1	15.3	15.1	45.5

Adding Links to Your Table

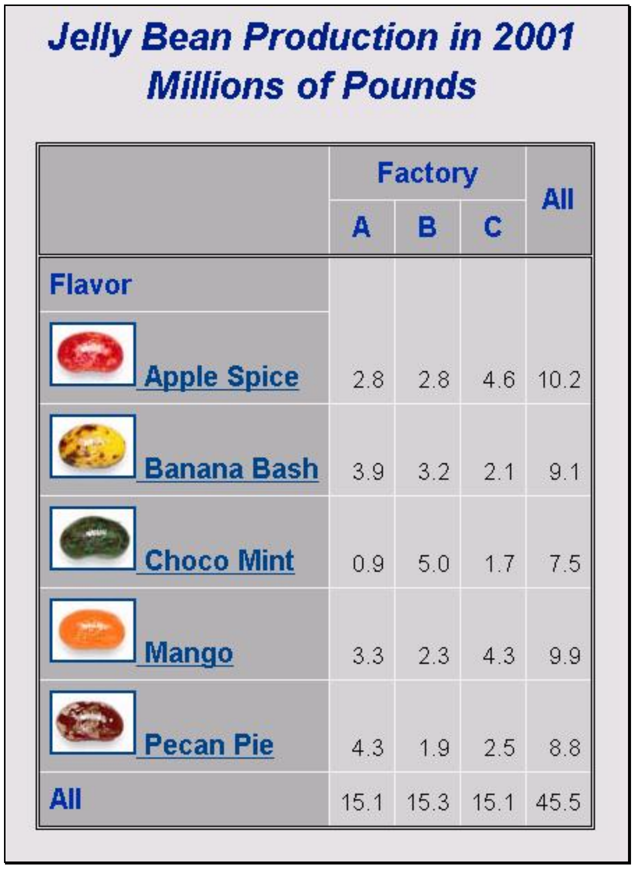
You can also add hyperlinks to your table by using the URL style element. Using the URL style element is similar to using the PREIMAGE style element. Create a format to assign file names, or URLs to the different levels of the class variable, then set the URL style attribute equal to this format.

```
ODS HTML BODY='c:\sugi\jellybean8.html';
PROC FORMAT;
   VALUE $link
        'P' = 'C:\sugi\pecanpie.html'
         'B' = 'C:\sugi\bananabash.html'
         'A' = 'C:\sugi\applespice.html'
         'M' = 'C:\sugi\mango.html'
         'C' = 'C:\sugi\chocomint.html';
```

```
PROC TABULATE DATA=production FORMAT=4.1;
   CLASS Factory Flavor;
   CLASSLEV Flavor/
      STYLE={PREIMAGE=$image. URL=$link.};
   VAR MPounds;
   FORMAT Flavor $flav.;
   TABLE Flavor ALL,Factory*SUM=''*MPounds=''
         ALL*SUM=''*MPounds='';
RUN;
ODS HTML CLOSE;
```

Table 12 shows the results of this program. You can see that the flavor names and also the images are now hyperlinks.

Table 12. TABULATE HTML output using the DEFAULT style and the URL and PREIMAGE attributes on the CLASSLEV statement.

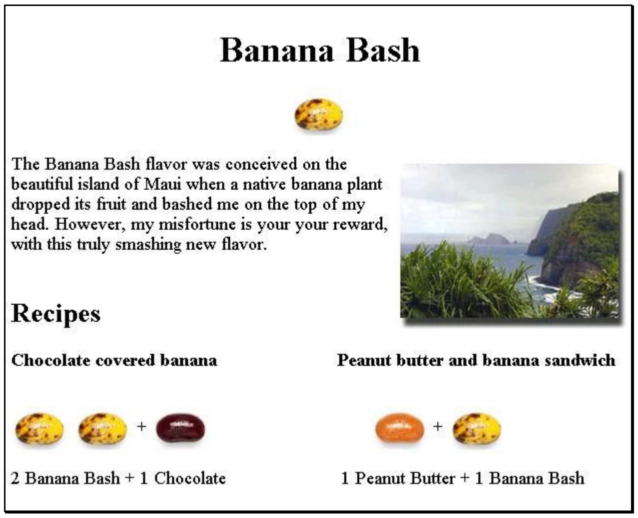


Jelly Bean Production in 2001
Millions of Pounds

	Factory			All
	A	B	C	
Flavor				
Apple Spice	2.8	2.8	4.6	10.2
Banana Bash	3.9	3.2	2.1	9.1
Choco Mint	0.9	5.0	1.7	7.5
Mango	3.3	2.3	4.3	9.9
Pecan Pie	4.3	1.9	2.5	8.8
All	15.1	15.3	15.1	45.5

Now when you display this page in your browser and you click on the words Banana Bash, or the image, then you will link to the page listed in the URL. In this case, we are linking to the static HTML page shown in Table 13.

Table 13. Static HTML page linked from Banana Bash in Table 12.



Banana Bash

The Banana Bash flavor was conceived on the beautiful island of Maui when a native banana plant dropped its fruit and bashed me on the top of my head. However, my misfortune is your your reward, with this truly smashing new flavor.

Recipes

Chocolate covered banana

2 Banana Bash + 1 Chocolate

Peanut butter and banana sandwich

1 Peanut Butter + 1 Banana Bash

Adding Dynamic Links

Each of the three Jelly World factories offers tours of their facilities, and to get an idea of how people like the new flavors they are producing, they give each visitor samples of the flavors and ask them to vote for their favorite. The data are stored in a SAS data set named Visitors, and sample observations are shown in Table 14.

Table 14. Selected observations from the Visitors SAS data set.

```
 Obs    Factory        Date    Visitor  Favorite

   1       A       01/02/01        1        P
   2       A       01/02/01        2        A
   3       A       01/02/01        3        C
   4       A       01/02/01        4        C
   5       A       01/02/01        5        P
   6       A       01/02/01        6        P
   7       A       01/02/01        7        B
   8       A       01/02/01        8        B
   9       A       01/02/01        9        B
  10       A       01/02/01       10        P
```

We want to add hyperlinks to our table which will link to reports showing the current vote tallies. Because the Visitors data set is being updated continually, we want to generate the reports on the fly.

Dynamic Web Architecture

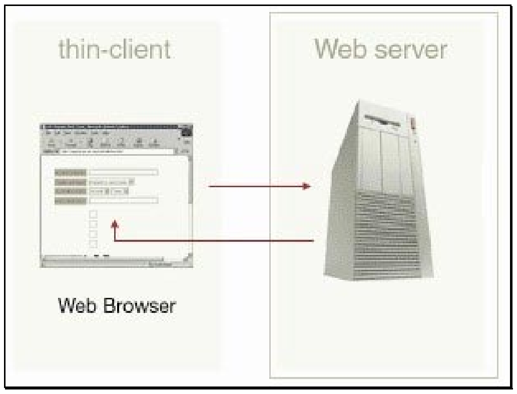
The examples described so far demonstrate how ODS can deliver information on the Jelly World factories via SAS programs being executed on your machine. The steps include:

1. Edit the SAS program
2. Execute the SAS program
3. Results are generated to an HTML file

ODS generates effective HTML output which can easily be reviewed on screen for large audiences. However, it requires the challenging step of running a SAS program since these reports are often delivered to users who do not know SAS or have it on their desktop. This is where ODS meets SAS/IntrNet to form an effective information delivery system.

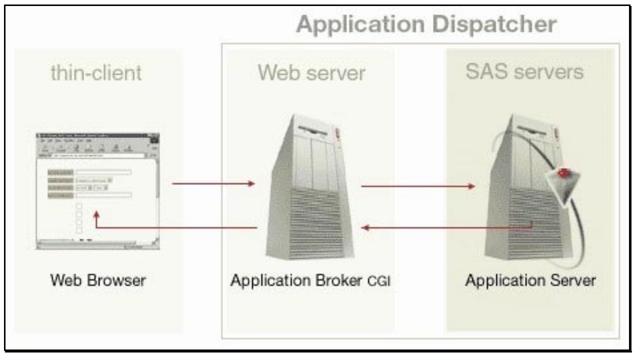
The simplest approach is for SAS programmers to execute SAS programs and place the resulting HTML ODS output onto the web server so that others can view it. This architecture is delivering static HTML pages similar to some SAS publishing tools. The user will view the page on the website as shown in Figure 1 below:

Figure 1. Static HTML delivery via web server.



This allows users to review SAS reports when they may not be able to otherwise. The reports, however, are static and are only updated as often as the SAS programmer performs the manual steps. In order to make this a truly dynamic request, SAS/IntrNet enables all users the ability to execute SAS programs themselves through the web browser and also review the resulting reports. The SAS/IntrNet architecture slightly changes the flow by adding the application server as shown in Figure 2 below.

Figure 2. SAS/IntrNet architecture.



The users do not need to be aware of the fact that there is another server working on their behalf. This will be transparent to them since the resulting HTML web page with the nice ODS HTML table is delivered through a simple click of a mouse. The difference is in the hyperlink. The dynamic SAS/IntrNet approach delivers the results through a hyperlink that is no longer pointing to a static HTML output produced from a program. Instead, the link points to a SAS program that is executed on-the-fly and then the resulting HTML report is delivered back to the browser.

The setup and configuration of the web server and the SAS/IntrNet application server is beyond the scope of this paper. We will assume that everything has been set up, and that the program can be accessed through a library configured by the SAS/IntrNet administrator.

Delivering ODS Programs with SAS/IntrNet

Most of the hard work is already done during the development of your ODS programs. The steps to make them available over the web dynamically using SAS/IntrNet are relatively easy. It requires the following steps:

1. Moving the ODS SAS programs onto the SAS application server in the designated directory
2. Updating the BODY option within the ODS Statement
3. Updating the user's HTML link with the new link which points to this program

Once the users click on this link, the SAS program will be executed at that time and the results will be delivered dynamically. This means that if the data is changing, they will get the most up-to-date information possible. The three steps above are described in more detail below.

Moving Programs onto Application Servers

The first step in making your ODS programs available to users via SAS/IntrNet is placing them into the right location. The SAS/IntrNet application server has predetermined locations where it can execute programs. This is defined by the administrator through the use of PROC APPSRV. The application server procedure needs to be programmed prior to the start of the application server so that it knows where to find your programs. An example of the code which defines the program location would look like:

```
proc appsrv port=5003 unsafe='&";%''';
  *** Define Program Location Example ***;
  allocate file sugi 'd:\sugi 27\saspgm';
  proglibs sugi;
run;
```

In this example, all programs copied to the application server in the location of `d:\sugi 27\saspgm`, will be available through the library name `sugi`. The administrator can assign as many of these as necessary to organize your dynamic programs.

Updating ODS Programs

Once your programs are copied to the proper location, you need to update them so that the output will be delivered across the web rather than to a physical file. In the previous examples, the BODY option points to an HTML file such as the one shown below:

```
ODS HTML BODY = 'c:\sugi\jellybean2.html';
```

This is where the output file would be created. Rather than a physical file on disk, the dynamic approach uses a system defined file reference _WEBOUT. The updated ODS statement would therefore look like:

```
ODS HTML BODY=_webout;
```

The _WEBOUT location is actually a pipe which delivers the HTML output across the web server and onto the user's browser as previously shown in Figure 2.

Updating the User's HTML Link

In the example of a static HTML page, the files are placed under a web server directory. The web server delivers them to users through a corresponding address referred to as a Universal Resource Locator or URL. An example URL is:

```
http://myserver/sugi/jellybean2.html
```

The users need to link to this URL in order to review the `jellybean2.html` report. The updated SAS/IntrNet approach links to the SAS program which is then executed to produce this report. An example of this is:

```
http://myserver/cgi-bin/broker.exe?
_SERVICE=default&_PROGRAM=sugi.jellybean.sas
```

There are actually four elements to this new URL that are different from the static version. The differences are as follows:

- The sub-directory is no longer `sugi` but rather `cgi-bin`.
- This is where SAS/IntrNet stores the application (broker.exe) to communicate with your SAS program.
- The name of the SAS/IntrNet service. The administrator can set up multiple services, but for our examples we'll use the default.
- The program is represented by a three level dot notation. These levels include:
    - `sugi` – file reference to program location
    - `jellybean` – program name
    - `sas` – program extension

This may seem rather complicated but it is something that the user does not even see. The result from the perspective of the browser is the same as any other hyperlink.

Adding Dynamic Hyperlinks through ODS

In the preceding example, ODS styles were used to create hyperlinks between one HTML file and another. That example was:

```
PROC TABULATE DATA=production FORMAT=4.1;
   CLASS Factory Flavor;
   CLASSLEV Flavor/
      STYLE={PREIMAGE=$image. URL=$link.};
   VAR MPounds;
   FORMAT Flavor $flav.;
   TABLE Flavor ALL,Factory*SUM=''*MPounds=''
         ALL*SUM=''*MPounds='';
RUN;
```

The `$link.` format specified the locations to static HTML pages. The same technique can be updated to have the result delivered as the output of your executed SAS program. This means that when the user clicks on the hyperlinks, SAS/IntrNet will actually execute your ODS program and return the HTML output. The key difference is in the hyperlink. In our previous example, links are defined as follows:

```
PROC FORMAT;
   VALUE $link
       'P' = 'C:\sugi\pecanpie.html'
       'B' = 'C:\sugi\bananabash.html'
       'A' = 'C:\sugi\applespice.html'
       'M' = 'C:\sugi\mango.html'
       'C' = 'C:\sugi\chocomint.html';
```

The update can be set to invoke SAS programs. Since the program name is the only thing that is different, a macro variable can be used to simplify the coding. Here is the updated example.

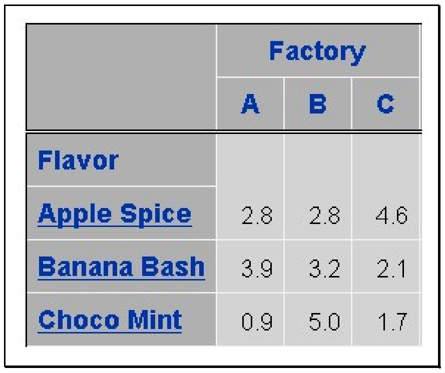
```
%let intrnet = http://myserver/cgi-bin/
broker.exe?_SERVICE=default&_PROGRAM=sugi;

PROC FORMAT;
   VALUE $link
       'P' = "&intrnet.pecanpie.sas"
       'B' = "&intrnet.bananabash.sas"
       'A' = "&intrnet.applespice.sas"
       'M' = "&intrnet.mango.sas"
       'C' = "&intrnet.chocomint.sas";
run;
```

In this example, each program can do whatever you want it to do. For example, the `pecanpie.sas` program can summarize the frequency of Pecan Pie by factory using PROC FREQ and perhaps `applespice.sas` can summarize Apple Spice with PROC MEANS. The choice is up to you. In general, the user will use the hyperlink as a drill-down to more information specific to that link.

Parameterized Links

Hyperlinks drill down to more information. In our example, the hyperlinks are categorized by flavor.

Table 15. Partial table of jelly bean production with hyperlinks by flavor.



	Factory		
	A	B	C
Flavor			
Apple Spice	2.8	2.8	4.6
Banana Bash	3.9	3.2	2.1
Choco Mint	0.9	5.0	1.7

Beside the fact that Apple Spice is a different flavor than Banana Bash, they are very similar from the standpoint of summary statistics. This implies that if we decide to use PROC FREQ as the next report to be generated for the drill down link of Apple Spice, it makes sense to also apply the same PROC FREQ program for Banana Bash. Rather than calling different programs that do the same thing, a parameter can be passed to the program through the link.

Parameters are separated by ampersand (&). For example, the following link sends two parameters to SAS/IntrNet including: service and program.

```
_SERVICE=default&_PROGRAM=sugi.jellybean.sas
```

In addition to these parameters, you can also add user-defined parameters. For example:

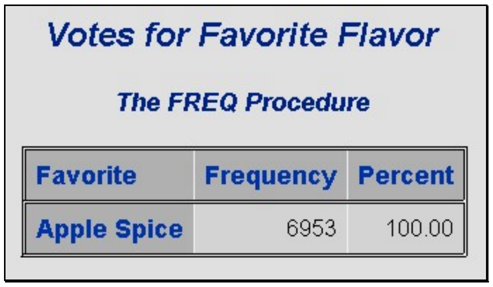
```
_PROGRAM=sugi.frequency.sas&flavor=A
```

In this example, SAS/IntrNet will define a macro variable named `flavor` with the value of 'A' and send it to the `frequency.sas` program. An example of the PROC FREQ program, which uses the `flavor` parameter is as follows:

```
ODS HTML BODY= _webout;

TITLE 'Votes for Favorite Flavor';
PROC FREQ DATA= visitors;
   FORMAT Favorite $flav.;
   WHERE favorite="&flavor";
   TABLES Favorite/NOCUM;
RUN;

ODS HTML CLOSE;
```

SAS/IntrNet delivers the parameter in the form of a macro variable so the `WHERE` condition in this case resolves to the flavor that was passed. In this example, if the selection was: `flavor=A`, the output would be:

Table 16. ODS PROC FREQ by Apple Spice flavor.



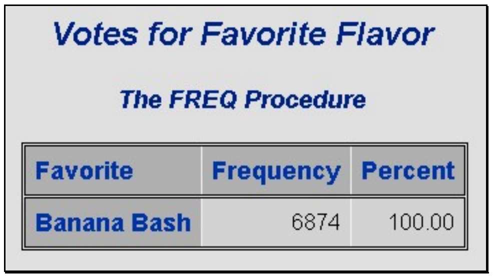
Votes for Favorite Flavor

The FREQ Procedure

Favorite	Frequency	Percent
Apple Spice	6953	100.00

On the other hand, if the user were to click on the Banana Bash hyperlink, the same program would execute, but a different parameter would be passed as `flavor=B`. In this case, the results would be:

Table 17. ODS PROC FREQ by Banana Bash Flavor.



Votes for Favorite Flavor

The FREQ Procedure

Favorite	Frequency	Percent
Banana Bash	6874	100.00

This is a much more effective way of developing SAS programs. Each program acts like a SAS macro and users can link to it whenever they need it.

Different Types of Links

Besides links being placed upon text, hyperlinks can also be placed upon images as shown in our previous example. In that example, the style option has the link placed upon the graphic image of the jelly bean.

```
PROC TABULATE DATA=production FORMAT=4.1;
   CLASS Factory Flavor;
   CLASSLEV Flavor/STYLE={PREIMAGE=$image.
      URL=$link.};
```

These same links can be converted to invoke interactive SAS programs just like their text counterparts. There are other possible links which SAS/IntrNet can deliver that go beyond the scope of this paper. However, a couple of brief descriptions can show you some of the possibilities:

- **Buttons:** HTML can also include buttons. The same frequency link could be represented as a button as shown here:

- **SAS/GRAPH:** The image source location of the jelly bean would have the HTML code of the following:

  ```
  <img src="images/jellybean.jpg">
  ```

  In this example, the jellybean is delivered through the web server in a directory named `images`. SAS/IntrNet can replace this source with a URL that calls a SAS/Graph program. The result is a dynamically generated SAS/Graph output.

These are some examples that allow SAS/IntrNet to go beyond simple static interfaces. It enables web-based tools to be full-fledged dynamic applications that can fit a wide variety of needs.

Conclusion

ODS makes it easy to convert existing programs into nice HTML output. The browser is a perfect tool for delivering these HTML pages to a large audience. The integration of SAS/IntrNet and ODS enables a new class of applications where each page is the result of a program which then links to another program. This tapestry of HTML pages with a wide-ranging selection of HTML hyperlinks creates a powerful platform for report and data delivery, as well as full-blown enterprise information systems.

References

Fehlner, William (1999). Making the Output Delivery System (ODS) Work for You. *Proceedings of the Twenty-fourth Annual SAS Users Group International Conference*. SAS Institute Inc., Cary, NC.

Olinger, Chris (2000). ODS for Dummies. *Proceedings of the Eighth Annual Western Users of SAS Software*. SAS Institute Inc., Cary, NC.

SAS Institute Inc. (1999). *SAS Procedures Guide, Version 8*. SAS Institute Inc., Cary, NC.

SAS Institute Inc. (1999). *The Complete Guide to the SAS® Output Delivery System, Version 8*. SAS Institute Inc., Cary, NC.

Slaughter, Susan and Lora D. Delwiche (2001). ODS for Reporting with Style. *Proceedings of the Ninth Annual Western Users of SAS Software*. SAS Institute Inc., Cary, NC, p. 636-645

About the Authors

Susan Slaughter and Lora Delwiche are authors of *The Little SAS Book: A Primer* published by SAS Institute. Sy Truong is a Systems Developer for Meta-Xceed, Inc. They may be contacted at:

Susan J. Slaughter	(530) 756-8434 susan@avocetsolutions.com
Sy Truong	(510) 713-1686 sy.truong@meta-x.com
Lora D. Delwiche	(530) 752-6285 llddelwiche@ucdavis.edu