Output Delivery System

The Basics *and* Beyond

Lauren E. Haworth
Cynthia L. Zender
Michele M. Burlew

SAS® Press

sas

Praise from the Experts

"Output Delivery System: The Basics and Beyond is a must-read for SAS users at all levels of experience. The authors certainly know their stuff, and their enthusiasm about the power and flexibility of ODS is evident throughout the book."

-Mike Rhoads
Westat

"Lauren Haworth, author of the previous pioneering book on ODS, has been joined by SAS's own Cynthia Zender, expert on all aspects of SAS reporting, and Michele Burlew, the versatile and prolific author of several other SAS books. With a team like this, SAS users could expect an outstanding resource, and we got it."

LeRoy Bessler PhD
SAS User Since 1978

"Written for the user who wants to learn by following examples, the presentation style is much less a reference manual and much more a guide to 'how to get the problem solved.'"

Art Carpenter
CALOXY

"This book is the *Joy of Cooking* for SAS ODS users. It gives good explanations for the ODS novice and is a complete resource for experienced users with questions. It is the book I've been waiting for."

Cecilia Mauldin
CDD

"I particularly like the authors' style, which is both informative and friendly, just like having a top-rate instructor giving you one-to-one training."

Peter Bennett
Director, I.T. Training Services Limited, UK

"…the format of code and explanation on one page and the resulting output on the facing page makes following the examples quite simple. The examples build on one another and by the end of a chapter the reader has a very good understanding of the topic covered. ODS is very powerful and can be complicated – this book really does take the reader through ODS, from the basics and beyond."

Pete Lund
Looking Glass Analytics

"I haven't enjoyed a technical book as much as I enjoyed reading this one. I've been impressed with the writing styles of all 3 authors, and this book is another easy read. What impressed me was the detail incorporated into every example, and, just when I had a burning question about one example, there was an example there immediately to answer it."

Philip R. Holland
Consultant and Author
Holland Numerics Ltd.

"If you want to expand your reporting capabilities multi-fold and avoid hours of

troubleshooting, this book is definitely worth a read!”

Christine Iodice
Statistical Programmer
Exelixis Inc

sas Publishing

Output Delivery System

The Basics *and* Beyond

Lauren E. Haworth
Cynthia L. Zender
Michele M. Burlew

THE POWER TO KNOW.

The correct bibliographic citation for this manual is as follows: Haworth, Lauren E., Zender, Cynthia L., and Burlew, Michele M. 2009. *Output Delivery System: The Basics and Beyond*. Cary, NC: SAS Institute Inc.

Output Delivery System: The Basics and Beyond

Copyright © 2009, SAS Institute Inc., Cary, NC, USA

ISBN 978-1-59994-660-3

ISBN 978-1-60764-383-8 (electronic book)

All rights reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a Web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

U.S. Government Restricted Rights Notice: Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19, Commercial Computer Software-Restricted Rights (June 1987).

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 2009

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

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 registered trademarks or trademarks of their respective companies.

Contents

Acknowledgments
About This Book
Part 1

Introduction

Chapter 1

Why Use ODS?

Limitations of SAS Listing Output

Difficulties with Importing Standard Listing Output into a Word Processor

ODS Output: A Variety of Formats to Fit Your Needs

Chapter 2

ODS Basics

Overview

Table Definitions and Style Definitions

Output Objects and Output Destinations

ODS References and Resources

Part 2

The Basics

Chapter 3

HTML Output

Generating Your First HTML File

Exploring the Body File: The Basics of HTML

Exploring the HTML Body File: Graphics

Combining the Results of Several Procedures

Generating a Table of Contents

Exploring the Frame File

Breaking Up Output into Separate Files

Simulating Page Breaks Using Cascading Style Sheets

More on Page Breaks Using the NEWFILE= Option

Changing Procedure Labels in the Table of Contents

Changing Multiple Procedure Labels

Changing the Title Bar

Creating Alternate HTML Formats

Creating Multiple Copies of the Same Output

Exploring Links and Anchors

Using Links and Anchors to Link Your Results to Your Web Site

Adding HTML Code to Customize Your Output
Limiting Inclusion of HTML Tags in Output
Applying Inline Formatting
Adding JavaScript to Your Output
Creating and Using Cascading Style Sheets
Creating Drill-Downs within Output Tables
Creating Output with Just the Results
Appending to an Existing HTML File
Appending to an Existing Table of Contents
Moving ODS Output into Your Internet or Intranet Site
Sending HTML Results Automatically via E-Mail
Importing Your ODS Output into a Spreadsheet
Using ODS Suboptions

Chapter 4

RTF Output

Creating Your First RTF File
Importing into a Word Processor
Working with Headers and Footers in Microsoft Word
Handling Page Breaks in RTF Output
Controlling Page Orientation
Controlling Page Size
Controlling Page Breaks
Controlling Page Margins
Creating Multiple Columns on a Page
Adding “Page X of Y” to RTF Reports
Adding Static or Dynamic Dates
Adding Customized Dates
Placing Title and Footnote Text in the Body of the Document
Adding Images to RTF Output
Adding Text
Adding More Complex Text
Adding Hyperlinks to RTF Output
Adding a Table of Contents to RTF Output
Completing the Table of Contents
Specifying Metadata in Your RTF File
Aligning Decimals in RTF Output
Exploring RTF as Created by Microsoft Word
Inserting Field Codes into Your RTF Output
Enhancing RTF Output by Using RTF Control Words and Field Codes
Sending RTF Field Codes and Control Strings Only to RTF Output
Importing RTF Output into a Slide Show

More Enhancement of RTF Output Using RTF Control Words
The New TAGSETS.RTF Destination
Inserting Text with the PREPAGE= Option
Using the TAGSETS.RTF OPTIONS Suboptions
Generating a Table of Contents with TAGSETS.RTF
Changing Table Headers with RTF Control Strings
Rotating or Aligning Table Headers with RTF Control Strings
Adding Hyperlinks to TAGSETS.RTF Output

Chapter 5

PDF Output

Creating a PDF File
Creating Bookmarks in Your PDF File
Modifying the Display of Bookmarks in Your PDF File
Creating PDF Files That Include Hyperlinks and Annotations
Another Means of Adding Hyperlinks and Annotations to PDF Output
Optimizing PDFs to Reduce Size and Increase Compatibility
Creating PDFs with Customized Metadata
Creating a Linked Set of PDF Results
Adding “Page X of Y” Numbers to PDF Output
Controlling Page Breaks
Sending ODS Output to Your Printer
Selecting a Printer
Creating a PostScript File from Your ODS Printer Output
Using the SAS Printer Drivers
Using the SAS PostScript Driver
Printing in Color
Creating Consistent Printed Pages
Creating Multiple Columns on a Page
Controlling Page Orientation
Controlling Page Margins
Controlling Page Size
Creating Multiple Versions of the Same Output
SAS System Options That Affect ODS Printer Output
Converting to PDF with Acrobat

Part 3

Output Structure

Chapter 6

Exploring ODS Output

The ODS TRACE Statement

Identifying Output Objects in Your Listing
Identifying Output Objects by Label Path
Viewing the Results
TRACE Examples: Other Procedures
TRACE Examples: Procedures without ODS Output
Chapter 7
Output Data Sets
Creating a Single Output Data Set
Comparing ODS Output Data Sets to Printed Output
Creating Multiple Output Data Sets from a Single Procedure
Creating Output Data Sets from Procedures with Complex Output
Creating Output Data Sets from Multiple Procedures: Part 1
Creating Output Data Sets from Multiple Procedures: Part 2
Creating Output Data Sets without Printed Output
Tip: Turning Off Output to the Results Window
Opening and Closing Output Data Sets
Using Data Set Options with Output Data Sets
Creating ODS Output Data Sets from the PRINT and REPORT Procedures
Chapter 8
Limiting ODS Output
Selecting a Subset of Your Results
Selecting Multiple Output Objects
Excluding the Output You Don't Want
Selecting Output for Multiple Procedures
Customizing Your Selections for Each Output Destination
Selecting Complex Combinations of Output Objects
Selecting with WHERE Conditions
Chapter 9
Using ODS in the DATA Step
Creating ODS Output in the DATAStep Using FILE and PUT Statements
Naming and Labeling the DATAStep Output Object
Selecting Variables to Include in the Output
Controlling the Layout of the Output
Labeling and Formatting the Columns of the Results
Modifying Output with the PUT Statement
Applying Inline Formatting
Using FILE PRINT Options and ODS
Chapter 10

Creating Custom Reports from the DATA Step

Understanding the Table Definition Used by the DATA Step
Creating a Custom Table Template with a DATA Step
Calculating Columns in a Custom Table Template
Creating Spanning Column Headers in a Custom Table Template
Creating a Custom HTML Table Definition
Performing Traffic Lighting in a Custom Table Template
Using Generic Columns in a Table Template
Using Macro Variables in a Custom Table Template
Using Dynamic Variables as Attribute Values in a Custom Table Template

Chapter 11

Modifying and Replaying ODS Output

The ODS DOCUMENT Destination
Using the Documents Window
Using PROC DOCUMENT to Manage Document Stores
Replaying a Document Store Using PROC DOCUMENT and Pop-Up Menus
Understanding the Fundamental Concepts of the ODS DOCUMENT Destination
Creating a New Document Store Interactively
Creating a New Document Store Interactively (continued)
Creating a New Document Store Interactively (continued)
Changing Titles and Page Breaks
Creating a New Document Store Using PROC DOCUMENT
Copying Objects into a New Document Store Using PROC DOCUMENT
Adding Titles and Correcting Page Breaks after Copying Output Objects to a New Document
Using the SASEDOC LIBNAME Engine
Summary of PROC DOCUMENT Statements

Chapter 12

Using Tagsets to Generate MARKUP Output

ODS MARKUP Destination
Creating CSV Files with ODS MARKUP
Creating HTML Result Files with ODS MARKUP
Creating XML Files with ODS MARKUP
Learning More about ODS MARKUP Tagsets
Modifying Tagset Events
Creating a Custom Tagset
Using a Custom Tagset Template
More about Tagset Templates
More about Using the Doc="Help" Suboption

Part 4

Output Style

Chapter 13

Style Template Definitions

Viewing a List of Available Style Template Definitions

Changing Style Template Definitions

Examining the Components of an ODS Style Template Definition

Modifying a Style Template Definition: A Simple Example

Finding Your Custom Style Template Definitions

Chapter 14

Understanding the Built-In Style Template Definitions

Understanding the Default Style in SAS 9.1.3

Understanding the Default Style in SAS 9.2

Understanding the Default Style in SAS 9.2 (continued)

Understanding the Color Style Elements

Understanding the Font Style Elements

Understanding the HTML and Text Style Elements

Understanding the Abstract Style Elements

Changing Specific Style Elements

Startup and Shutdown Elements

Changing HTML Frames, Indexes, and Tables of Contents

Changing HTML Body Style Attributes

Changing Table Style Elements (Part 1 of 2)

Changing Table Style Elements (Part 2 of 2)

Changing Header-Related and Data Cell-Related Style Elements (Part 1 of 2)

Changing Header-Related and Data Cell-Related Style Elements (Part 2 of 2)

Adding Images to Style Elements

Modifying Titles and Footnotes Using Style Elements (Part 1 of 2)

Modifying Titles and Footnotes Using Style Elements (Part 2 of 2)

Understanding the Differences in the ODS Printer Style

Understanding Differences in the RTF Style

Chapter 15

Modifying Output Fonts

Identifying the Font Definitions for Each Part of Your Output

Modifying the Default Font Sizes for Printer Output

Modifying the Default Font Sizes for RTF Output

Modifying the Default Typefaces

Identifying Available Fonts

Modifying the Default Font Weights, Font Styles, and Font Widths

Creating a New Font Definition
Modifying Fonts for Procedures That Use Fixed-Width Fonts
Chapter 16
Modifying Output Structure
Modifying the Table Borders
Removing the Table Borders and Rules
Removing Table Borders and Rules from Printer Output
Changing the Spacing Inside Tables
Removing Numbers from the Table of Contents in HTML Output
Setting the Width for HTML Output
Controlling RTF Page Margins
Simulating Listing Output with FILE PRINT ODS
Hiding Procedure Titles
Chapter 17
Modifying Output Colors and Images
Exploring the Style Elements That Control Color
Modifying the Color of Your Titles
Identifying Colors
Modifying Background Colors
Modifying Colors in Tables
Adding a New Color to the Color Scheme
Designing a Template to Enhance Accessibility
Adding a Logo to Your Output
Adding a Logo and a Custom Heading to Your HTML Output
Appendix
Operating System Differences
Setting Up the Example for Windows
Revising the Example for UNIX (HPUX, Sun, Solaris, Linux, etc.)
Revising the Example for z/OS
Revising the Example for OpenVMS
Sources of Additional Information
Index
Adding a Background Image to Your HTML Output
Adding Images to RTF Output (with Templates)
Adding Images to Each Page of Your RTF Output
Adding Images to Printer Output (with Templates)
Chapter 18

Doing More with Styles and ODS

Removing Access to a Personal or Group Template

Chapter 19

Special Cases: The REPORT, TABULATE, and PRINT Procedures

Changing PROC REPORT Column Heading Style Attributes

Changing PROC REPORT Table Cell Style Attributes

Changing PROC REPORT Row Style Attributes

Changing PROC REPORT Subtotals and Totals Style Attributes

Adding Traffic Lighting to PROC REPORT Table Values

Adding Graphics to PROC REPORT Tables

Creating an Output Data Set from PROC REPORT

Controlling Column Widths with PROC REPORT

Inserting Blank Lines with PROC REPORT

Creating Cleaner PROC REPORT Output Using SPANROWS

Changing PROC TABULATE Row and Column Heading Style Attributes

Changing PROC TABULATE BOX= Option Formatting

Changing PROC TABULATE Table Cell Style Attributes

Changing PROC TABULATE Row Style Attributes

Changing PROC TABULATE Subtotals and Totals Style Attributes

Adding Traffic Lighting to PROC TABULATE Table Values

Adding Graphics to PROC TABULATE Tables

Creating an Output Data Set from PROC TABULATE

Controlling Column Widths with PROC TABULATE

Changing PROC PRINT Column Heading Style Attributes

Changing PROC PRINT Table Cell Style Attributes

Changing PROC PRINT Style Attributes for Totals

Indenting Values with Margin Attributes Using LEFTMARGIN=

Indenting Values with Margin Attributes

Style Attributes for the REPORT, TABULATE, and PRINT Procedures

Chapter 20

Understanding ODS Graphics Output

Understanding Graphics Creation in SAS 9.2

Generating Graphics Output from SAS Procedures Using the ODS GRAPHICS Statement

Creating Graphic Images with the ODS Statistical Graphics Framework

Creating Graphics Output with the Statistical Graphics (SG) Procedures

Creating Output Using Graph Templates and the SGRENDER Procedure

Creating HTML Output with SAS/GRAPH

Using ODS Options and SAS/GRAPH Options to Change Output

Understanding Devices and Destinations

More about Devices and Destinations

Creating HTML Output: Graphs as GIFs or JPEGs
Creating HTML Output: Graphs as Java
Creating HTML Output: Graphs as ActiveX
Creating RTF Output: Graphs as ActiveX
Combining Graphs with Text Output
Additional SAS/GRAPH and ODS Resources

Acknowledgments

This book would not have been possible without the support and hard work of a number of people. First and foremost, we'd like to thank our spouses Steven, Lee, and John for their moral support. They graciously tolerated a lot of late nights and weekends spent by the authors muttering over programs, polishing this text, and reading page proofs.

At SAS, Julie Platt and John West were invaluable in providing constant encouragement and handling all of the details that go into publishing a book. In addition, we'd like to thank Eric Gebhart, Wayne Hester, Scott Huntley, David Kelley, Sandy McNeill, and Kevin Smith. Their efforts to keep us up-to-date with the rapidly growing features of the Output Delivery System were greatly appreciated. Our reviewers —Claire Bonney, Allison Booth, Paige Daniels, Troy Davis, Dan Heath, Wayne Hester, Anita Hillhouse, Tim Hunter, Scott Huntley, David Kelley, Bari Lawhorn, Lynn MacKay, Lelia McConnell, Kathryn McLawhorn, Chevell Parker, and Kevin Smith—generously provided us with their feedback and suggestions, which made this book a better product. We also want to thank the following people from SAS Publications: Joel Byrd, Patrice Cherry, Jennifer Dilley, Candy Farrell, Shelly Goodin, Stacey Hamilton, and Mary Beth Steinbach, for all of their contributions toward getting this book into your hands.

Finally, we'd like to thank the SAS user community for all of their suggestions. In particular, the SAS-L contributors and the SAS students gave us a number of ideas about topics to address.

About This Book

Purpose

Output Delivery System: The Basics and Beyond will teach you how to use a wide array of ODS techniques through a series of step-by-step examples. These techniques can transform your basic SAS output into customized reports; tailor your output for specific destinations such as HTML, word processors, and PDF; and enable you to select only the output that you need.

Is This Book for You?

This book is aimed at all levels of SAS users from all disciplines. Basic knowledge of SAS programming is necessary to understand the examples in this book. You should know how to write a DATA step and how to specify procedures. The most common Base SAS procedures used are PRINT, REPORT, and TABULATE. A few examples illustrate ODS features with SAS/STAT and SAS/GRAPH procedures, but a detailed knowledge of these products is not necessary to understand the ODS concepts being presented.

Some parts of the book might be more helpful than others, depending on your level of expertise. Here's a guide to what parts of the book will be the most help for you.

Beginning Users

If you are new to SAS and have little or no experience with ODS, you should read Part 1, "Introduction," for an overview of ODS and read Part 2, "The Basics," in detail. Concentrate on the material for the output destination that you're most likely to use first. After that, Parts 3 and 4, "Output Structure" and "Output Style," will be more understandable, and you can use them as needed.

Intermediate Users

If you have already created simple ODS output, you can skim Parts 1 and 2, "Introduction" and "The Basics," and move on to read Parts 3 and 4, "Output Structure" and "Output Style."

Advanced Users

If you are already comfortable with ODS output production, you will want to skim all parts of the book to find features you didn't know about. Spend more time reading in detail Parts 3 and 4, "Output Structure" and "Output Style."

All Users

The Appendix, "Operating System Differences," is a must-read for anyone not using the Windows platform.

Scope of This Book

This book covers ODS programming concepts from the beginning level to the advanced level. It is both an updated revision of Lauren Haworth's book, *Output Delivery System: The Basics*, and a new resource for examples of greater complexity. By adapting the examples in this book to your own data, you will learn to take basic SAS output and transform it into an HTML page, a word-processor-friendly RTF file, or printer-friendly PDF output. You will also learn how to generate table of contents pages and bookmarks in PDF files, and how to customize page numbering, headers, and footers in your RTF and PDF output.

Besides showing how to customize your output, this book shows you how you can use ODS to control how much output you get from a procedure and how to save output objects from procedures in data sets. The book explains how to use built-in ODS style templates and how to adapt them for your own needs. Through examples, you will learn how to identify the statements in a template that affect specific parts of your reports. The more advanced topics will show you how to use tagset templates to generate custom markup language tags and output. The book also introduces you to the ODS Statistical Graphics framework and the Statistical Graphics (SG) procedures.

How to Use the Examples

You don't have to read this book cover to cover to take advantage of the Output Delivery System (ODS) techniques that it contains. The book is set up as a series of examples so that you can zoom in on the information that best meets your needs.

If you flip through the book, you will see that the description of each example and the SAS code are always on the left-hand page, and the corresponding example output is always on the facing right-hand page. Everything you need to understand each section is right in front of you so you don't have to turn any pages. When you need help creating a particular type of ODS output, follow these steps:

- Find the chapter that covers the general type of output that you're interested in. For example, if you need help with HTML output, try Chapter 3, "HTML Output." Alternatively, if you need help with a particular ODS language statement, check the index to find pages with references to that statement.
- Flip through the chapter or examples and look at the output until you find an example that looks roughly like the output you're trying to create.
- After reading the explanation, use the example code to start creating your own output. Substitute your variables and data sets for the variables and data sets in the example code. If the example uses different SAS procedures than you need, figure out which procedures you want to use, and substitute them for the procedures in the example code.
- Do a test run of your code to see if you have the approximate result you were looking for. If you run into problems, reread the explanation.

After the basic structure is correct, you can work on cleaning up and refining your output. Part 4, "Output Style," covers topics related to revising the appearance of your output.

The Code in This Book

A great deal of code is presented in this book. Some of it is ordinary SAS procedures and DATA steps. In general, this code is not explained. Consult the SAS OnlineDoc for more information. The focus of the explanation in this book is on the ODS code.

In the code examples in this book, the ODS code is mixed in. To make it easier to spot the ODS code, all ODS statements are displayed in UPPERCASE. The rest of the code is in lowercase or mixed case. Do not be confused by this convention. ODS statements or options are equally correct whether they appear in uppercase, lowercase, or mixed case.

In general, all of the code in this book uses the default SAS options settings. However, the setting for DATE was changed from the default to NODATE. This was done to make the output easier to display in the book. It has little or no impact on the results themselves. If you want your output to match the output in the book, add the statement OPTIONS NODATE at the top of your code.

Software Used to Develop This Book's Content

This book is based on SAS 9.2 software. Although every effort has been made to include the latest information available at the time of printing, new features will be made available in later releases. Be sure to check out the SAS Web site for current updates, and check the SAS OnlineDoc for enhancements and changes in new releases of SAS.

SAS 9.2 saw some significant changes in how SAS creates ODS style templates. SAS 9.1.3 users can find on the author pages for this book equivalent 9.1.3 code for examples that are coded differently in SAS 9.1.3. See http://support.sas.com/authors.

Data Sets and Programs Used in This Book

This book uses several simple data sets to illustrate the concepts. You can access the example programs by linking to the author pages for this book at http://support.sas.com/authors. From this Web site, select "Example Code and Data" to display the SAS programs that are included in the book.

For an alphabetical listing of SAS books for which example code is available, see support.sas.com/bookcode. Select a title to display the book's example code.

If you are unable to access the code through the Web site, send e-mail to saspress@sas.com.

Tips and Warnings

Throughout the text, you will see tips and warnings related to some of the examples. They have distinctive icons to set them off from the rest of the text. Be sure to read any warnings associated with the examples before you try the code in your own programs.

This is the symbol used to identify a tip. Tips generally contain additional information about the example, or explain an alternate technique that can be used to create the example table.

This is the symbol used to identify a warning. Warnings generally contain additional information about potential risks associated with the technique used in the example.

SAS Author Pages

You can access the author pages for this book at http://support.sas.com/authors.

These pages include several features that relate to this specific book, including more information about the book and authors, book reviews, and book updates; book extras such as example code and data; and contact information for the authors and SAS Press.

Additional Resources

SAS offers you a wide variety of resources to help build your SAS skills and explore and apply the full power of SAS software. Whether you are in a professional or academic setting, there are learning products that can help you maximize your investment in SAS.

Bookstore	http://support.sas.com/publishing/
Training	http://support.sas.com/training/
Certification	http://support.sas.com/certify/
Higher Education Resources	http://support.sas.com/learn/
SAS OnDemand for Academics	http://support.sas.com/ondemand/
Knowledge Base	http://support.sas.com/resources/
Support	http://support.sas.com/techsup/
Learning Center	http://support.sas.com/learn/
Community	http://support.sas.com/community/

Comments or Questions?

If you have comments or questions about this book, you can contact the authors through SAS as follows:

Mail:

SAS Institute Inc.
SAS Press
Attn: <Author's name>
SAS Campus Drive
Cary, NC 27513

E-mail:

saspress@sas.com

Fax:

(919) 677-4444

Please include the title of the book in your correspondence.

For a complete list of books that are available through SAS Press, visit support.sas.com/publishing.

SAS Publishing News

Receive up-to-date information about all new SAS publications via e-mail by subscribing to the SAS Publishing News monthly eNewsletter. Visit support.sas.com/subscribe.

Part 1
Introduction

Chapter 1 Why Use ODS? 3
Chapter 2 ODS Basics 13

Chapter 1
Why Use ODS?

If all you want are quick results displayed to the screen or roughly printed out in a stack of output, you can put this book away right now.

However, if you're like the rest of us, you want more from your output. That's because most SAS output is shared with others.

You want your output to be appealing, to help you tell a story. You want your output to be impressive: it should look like you worked on it for hours, even if it took you only a few minutes. In addition, you want your output to be easy to share with other software applications and other users, whether they are across the hall from your office or across the globe.

This chapter shows how you can take some basic SAS output and transform it from a simple line-printer output into a beautiful integrated Web page, a word processor–friendly RTF file, high-resolution printed output, and a bookmarked PDF file.

This chapter does not show any Output Delivery System (ODS) code, but rather just gives you a taste of the types of output you can produce and what each type looks like. The following chapters get into the detail of the code.

Limitations of SAS Listing Output

Before you can appreciate the wonder of ODS, you need to see where we're starting from. The series of simple procedures in the following example describe some sales data. There's a PROC FREQ step, a PROC UNIVARIATE step, and a PROC PRINT step that lists some outliers in the sales data.

```
      options formdlim=' ';
title "Frequencies";
proc freq data=sales;
   table ProductName Department;
run;
title "Univariate";
proc univariate data=sales;
   var ItemsSold;
run;
title "Outliers for Price";
proc print data=sales;
   where ItemPrice>13;
   var ItemPrice;
run;
```

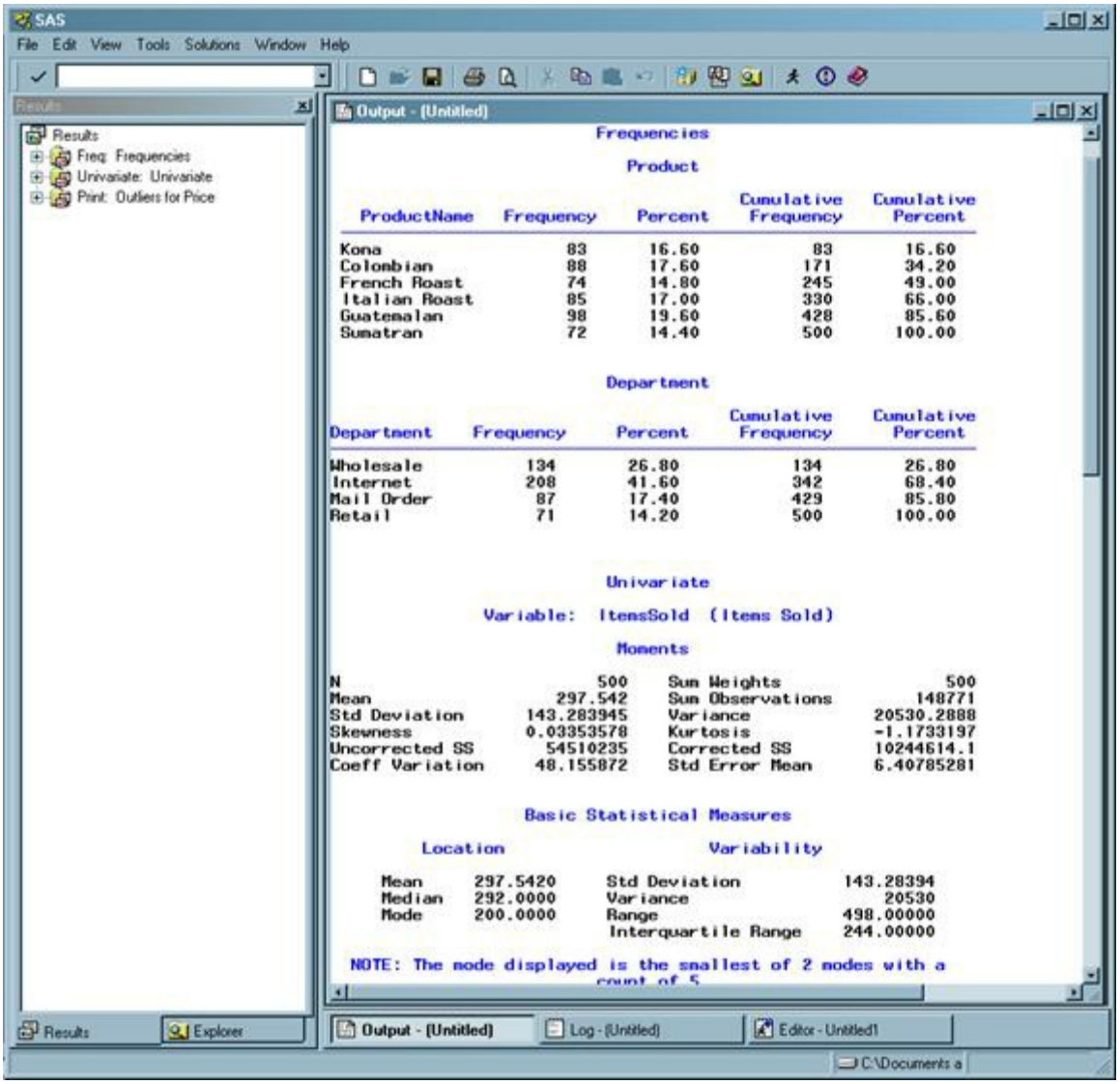
The procedures have been run in SAS 9.2, producing Figure.1.1. Because the program processes three different procedures, the resulting output is a series of three reports. The first report has two frequency tables, the second is a report with univariate statistics, and the third report shows line item details for the outliers. (Some of this output has been trimmed in order to fit the results on the page.) Figure 1.1 looks the same way in SAS®9 as it did back in SAS 6, or even SAS 5!

By default, SAS puts each procedure's results on a separate page. To keep this output as short as possible, the program specifies the FORMDLIM= option to remove the page breaks. However, even without the page breaks, the resulting output sprawls across a number of pages. In Figure 1.1, only the frequency tables and the first part of the univariate results fit on the first page.

If a reader of this report has time only to glance quickly at the top page, this output doesn't tell much of a story. You can't see all of the results. For example, what if the outliers for ItemsSold and ItemPrice are the most important message of the report? This information is buried on the second page. The extreme values for ItemsSold and the outlier report for ItemPrice can't be seen.

Also, this report is just plain ugly. Because this is SAS listing output, spaces are used to line up the columns of data and headings. This forces you to use a monospace font, rather than a proportional font. Proportional fonts (like the Times New Roman used for the text you are reading right now) are much easier to read. But if you used a proportional font with SAS listing output, nothing would line up. To see what SAS listing output would look like with a proportional font, look at Figure 1.2 on page 7.

Figure 1.1



(*Remaining output not shown in order to fit results on this page.*)

Difficulties with Importing Standard Listing Output into a Word Processor

When you're delivering a set of results, frequently you need to combine them with a text-based report that explains what is going on. Ideally, you'd like to combine the results and the text into a single document.

With SAS listing output, it's possible to import your results into a word processor and add explanations and annotations. However, it takes a bit of work to make the results look attractive.

Figure 1.2 shows what the output from the previous example looks like when it is opened in Microsoft Word. Because the default font in Word was set to Arial, a proportional font, these results look terrible. Nothing lines up. In addition, the lines under the table headings have been converted to a bunch of strange characters. This is because these lines were originally rendered in the SAS Monospace font. The codes that generate the smooth lines in table headings work just great if you stick to this font, but as soon as you switch fonts, you get nonsense characters.

To fix this output, the font can be changed to SAS Monospace. Now, if you look at Figure 1.3, you can see that everything lines up properly. However, it's still not very attractive. Monospace fonts are not as easy on the eye as proportional fonts. Table 1.1 illustrates some common proportional and fixed fonts, and shows the difference the proportional spacing can make. Not only are the proportional fonts more appealing, but they also save space. Notice how much longer the lines of fixed-width fonts have to be to display the same text.

Table 1.1 Common Proportional and Fixed Fonts

FONT	SAMPLE TEXT	TYPE
Arial	Sample text to show the appearance of this font.	Proportional
Times New Roman	Sample text to show the appearance of this font.	Proportional
Courier	Sample text to show the appearance of this font.	Fixed
SAS Monospace	Sample text to show the appearance of this font.	Fixed

If you want to use a more readable proportional font yet still keep your columns straight, you need to convert this output into Word tables. Then you can change to any font and everything will still line up.

Word is good at converting text to tables, but SAS listing output presents a challenge. Because the items are separated by spaces, Word creates a new table column wherever

spaces occur. This means that if you've got a space within a column (for example, between the words "French" and "Roast" in the row labels of the PROC FREQ), Word will split this column into two columns.

Although you can certainly use a word processor to clean up your SAS listing output, it's a lot of work because it requires so much tweaking to get things to look right. With ODS, there's no need to go through all this. Turn the page to enter the world of ODS.

Figure 1.2: SAS Listing Output after Importing into Word

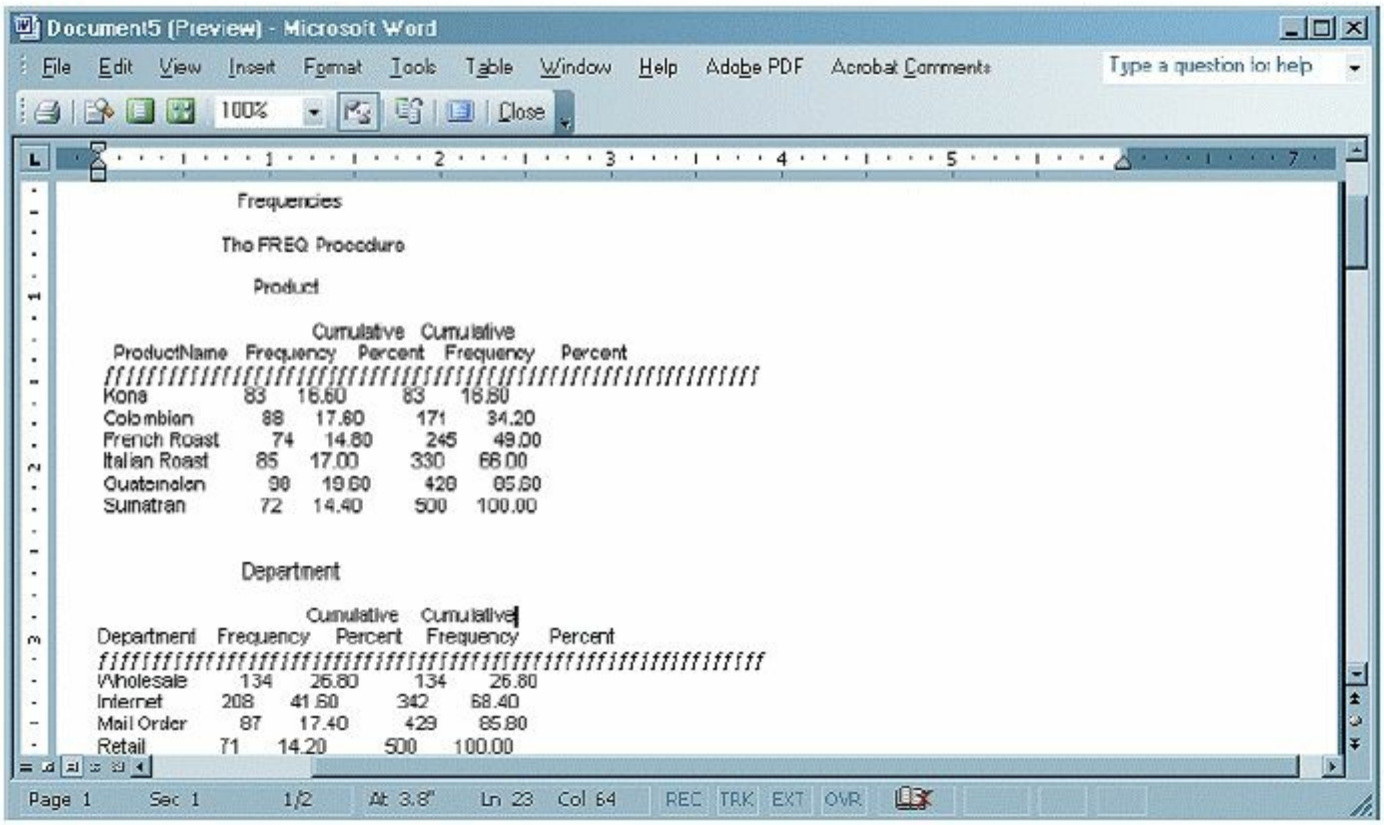
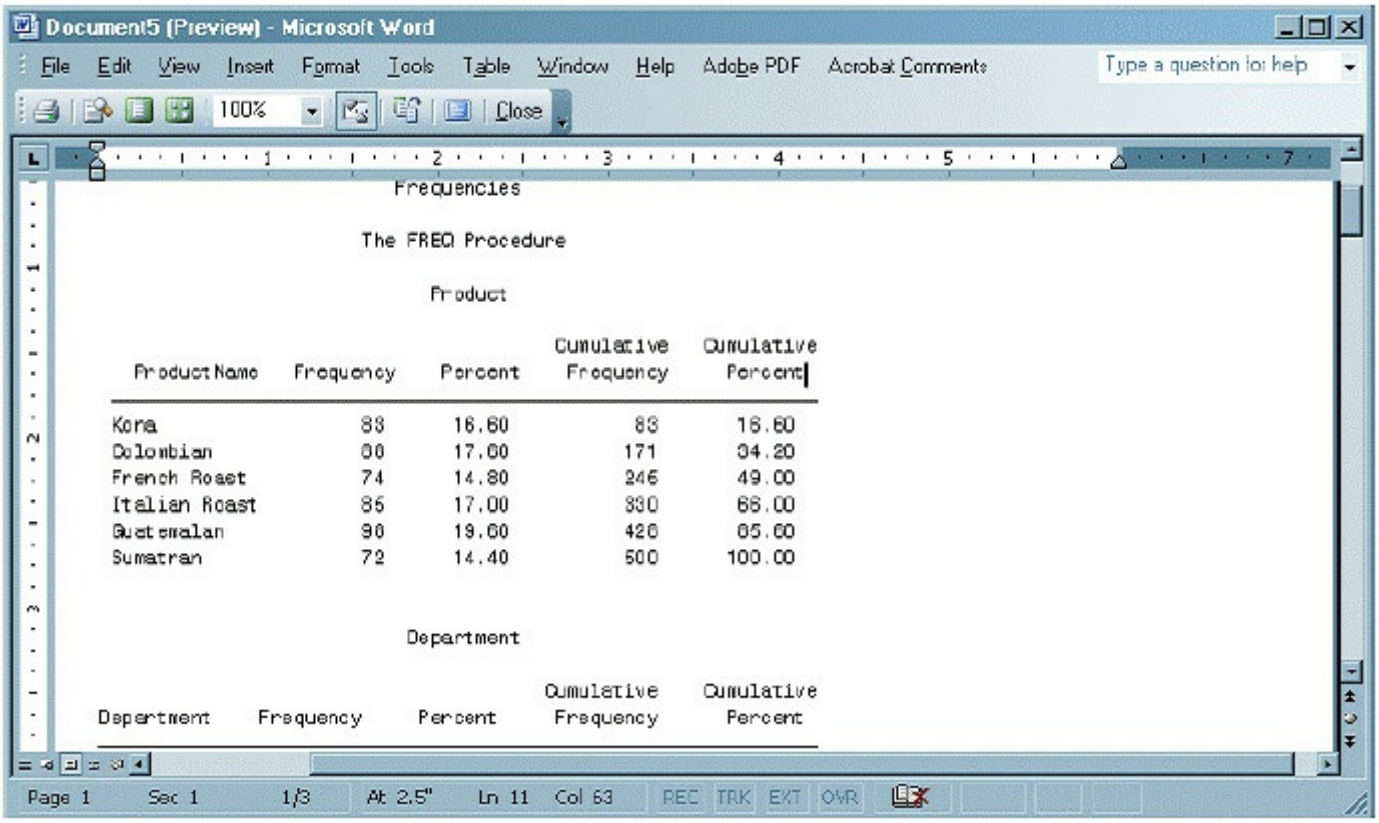


Figure 1.3: Results of Changing to a Non-Proportional Font



Frequencies

The FREQ Procedure

Product

Product Name	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Kona	83	16.60	83	16.60
Colombian	88	17.60	171	34.20
French Roast	74	14.80	246	49.00
Italian Roast	85	17.00	330	66.00
Guatemalan	96	19.60	426	85.60
Sumatran	72	14.40	500	100.00

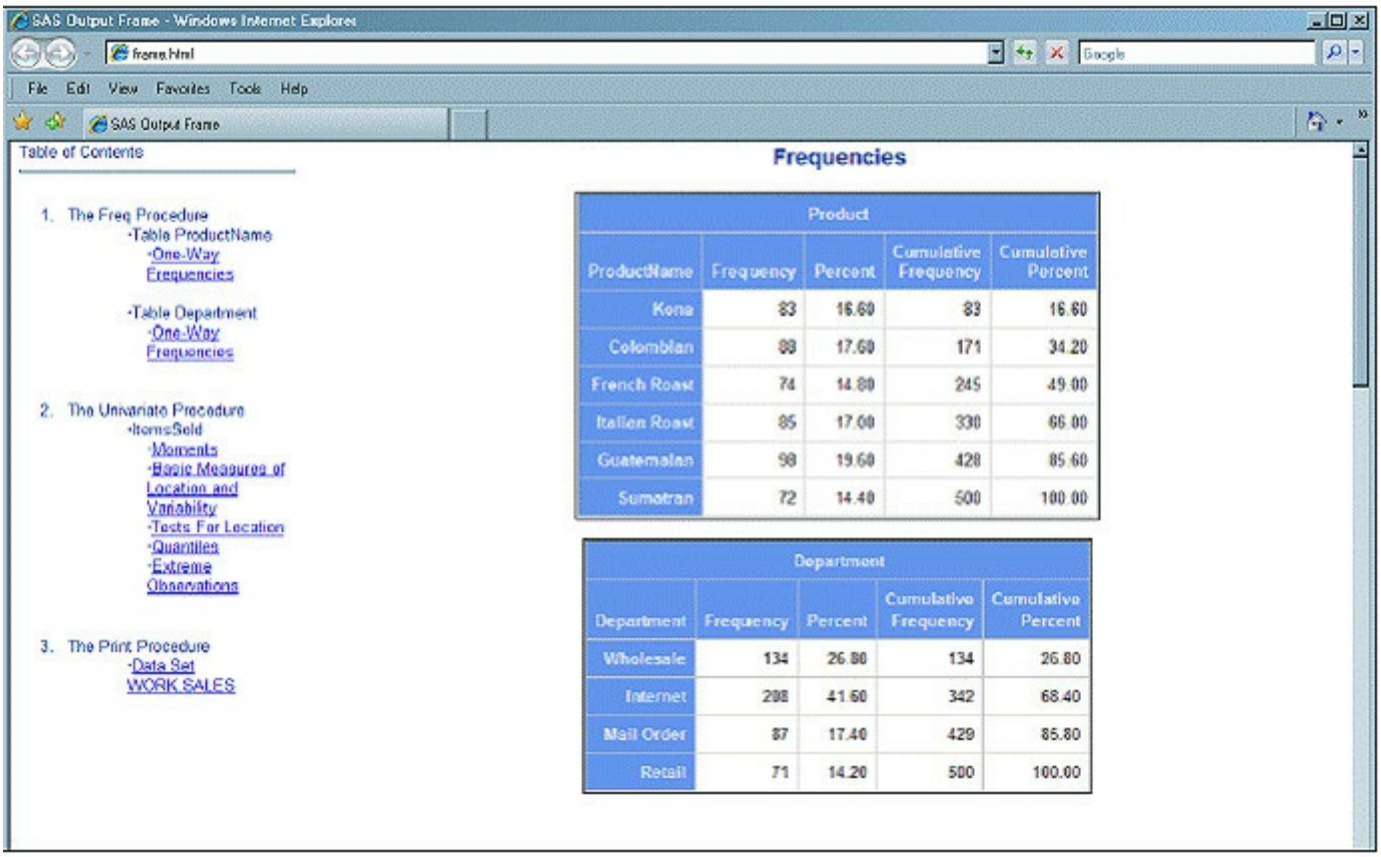
Department

Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent

ODS Output: A Variety of Formats to Fit Your Needs

The same SAS code as the previous example produces the output in Figure 1.4, but now ODS has directed the output to HTML. You are looking at how the output would appear to someone viewing it with a Web browser. Instead of using the default gray and blue color scheme, this output was produced with the Sasweb style.

Figure 1.4



Product				
ProductName	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Kona	83	16.60	83	16.60
Colombian	88	17.60	171	34.20
French Roast	74	14.80	245	49.00
Italian Roast	85	17.00	330	66.00
Guatemalan	98	19.60	428	85.60
Sumatran	72	14.40	500	100.00

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Wholesale	134	26.80	134	26.80
Internet	208	41.60	342	68.40
Mail Order	87	17.40	429	85.80
Retail	71	14.20	500	100.00

Instead of sprawling from page to page, as with the listing output, this output is neatly packaged with a table of contents to take you directly to whatever you want to view. Instead of using old-fashioned fixed-width fonts, the HTML output uses several good-looking fonts. Instead of being in black and white, the HTML output uses a style of your choice.

All of this adds up to a packaged set of results that you wouldn't be embarrassed to share with your boss. In addition, because this is HTML output, you can place it on your company Web site so your boss can view the report even if he or she is out of the country.

But why stop with HTML output? With ODS you can create the same report in a number of formats sure to please every user. The following example shows the same results, this time directed to RTF and opened in Word. The RTF and PDF versions of the file can even have a table of contents.

Figure 1.5

Product				
ProductName	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Kona	83	16.60	83	16.60
Colombian	88	17.60	171	34.20
French Roast	74	14.80	245	49.00
Italian Roast	85	17.00	330	66.00
Guatemalan	98	19.60	428	85.60
Sumatran	72	14.40	500	100.00

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Wholesale	134	26.80	134	26.80
Internet	208	41.60	342	68.40
Mail Order	87	17.40	429	85.80
Retail	71	14.20	500	100.00

Note the striking difference between this output and the results we got when we imported SAS listing output into Word in the previous example. The ODS output has been converted into Word tables and is ready to go with no tweaking needed.

If you wanted to change the font used for this output, you could do so without losing the column alignments. Now that your output is in a word processor, it's easy to add explanatory text and annotations. For example, if you wanted to explain something about the mail order figure, you could use a Word callout box to add a note with an arrow pointing to the number it describes.

The RTF output is best if you need to incorporate other information, or if you plan to send your output to others for further editing or annotation. But if all you need is an elegant printed report, you can use ODS to send high-resolution output directly to your printer. The following output shows the same results as they appear when printed using the ODS PRINTER destination.

Figure 1.6

Product				
ProductName	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Kona	83	16.60	83	16.60
Colombian	88	17.60	171	34.20
French Roast	74	14.80	245	49.00
Italian Roast	85	17.00	330	66.00
Guatemalan	98	19.60	428	85.60
Sumatran	72	14.40	500	100.00

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Wholesale	134	26.80	134	26.80
Internet	208	41.60	342	68.40
Mail Order	87	17.40	429	85.80
Retail	71	14.20	500	100.00

In addition, if you want to create output that you can put on a Web site, and that can be opened with Adobe Reader, you can also use ODS to create PDF files. If your goal is to create the most professional looking results from your output, Printer-family output is the way to go. You get crisp table lines and fonts, and you can use color if you have a color printer.

ODS Printer-family output also gives you the option of creating a PostScript or Printer Control Language (PCL) file, in addition to a PDF file, so you can make these printable files available in a variety of formats. If you expect visitors to your Web site will want to print out your results, PDF format is the best approach. HTML output looks great, and loads quickly for people with slower Internet connections, but printing capabilities are limited to browser printing interfaces.

There's one more type of ODS output we haven't seen yet. The previous examples all showed ways of producing reports from your output. But what if you want to do further analysis? In the past, only selected SAS procedures were able to produce output data sets that contained some of their results. ODS, however, enables you to create a data set from most procedures. The data set contains all of the results the procedure is capable of producing in printed output. In addition, ODS produces both the formatted and unformatted values of the variables. In this example, these values are the same, but in other cases, they might not be.

Though you can create data sets from any part of the output, the frequencies and the extreme observations from the univariate analysis were chosen in this example. Figure 1.7

shows the output of a data set created by ODS and PROC FREQ as presented with Viewtable.

Figure 1.7

	Table	ProductName	Product	Frequency	Percent	Cumulative Frequency	Cumulative Percent	Department	Department
1	Table ProductName	Kona	Kona	83	16.60	83	16.60		.
2	Table ProductName	Colombian	Colombian	88	17.60	171	34.20		.
3	Table ProductName	French Roast	French Roast	74	14.80	245	49.00		.
4	Table ProductName	Italian Roast	Italian Roast	85	17.00	330	66.00		.
5	Table ProductName	Guatemalan	Guatemalan	98	19.60	428	85.60		.
6	Table ProductName	Sumatran	Sumatran	72	14.40	500	100.00		.
7	Table Department		.	134	26.80	134	26.80	Wholesale	Wholesale
8	Table Department		.	208	41.60	342	68.40	Internet	Internet
9	Table Department		.	87	17.40	429	85.80	Mail Order	Mail Order
10	Table Department		.	71	14.20	500	100.00	Retail	Retail

Figure 1.8 shows the output of a data set created by ODS and PROC UNIVARIATE as presented with Viewtable.

Figure 1.8

	VarName	Lowest Value	Observation Number of Lowest Value	Highest Value	Observation Number of Highest Value
1	ItemsSold	51	48	547	154
2	ItemsSold	52	264	547	392
3	ItemsSold	52	143	548	142
4	ItemsSold	52	58	549	66
5	ItemsSold	54	440	549	127

If you compare the data set from the frequency distributions to the output shown on the previous page, you can see that the output data set contains every number that was in the table. With these data sets, you can now run further analyses using SAS procedures or the DATA step, or you can create summarized reports with selected results.

Chapter 2
ODS Basics

ODS is no longer a brand new tool. It is well documented and pretty easy to use. However, if you're new to ODS, you need to learn some terminology that is distinct from some of the SAS programming terminology you already know. This chapter explains these new terms and goes over the basics of what ODS does.

You could skip this chapter and go straight on to creating ODS output in Chapters 3 through 6. However, understanding the basics of how ODS works and knowing the terminology for its components will make the following chapters more understandable.

This chapter provides a brief overview of ODS terminology, followed by a simple syntax summary. You might want to bookmark the syntax summary to use as a reference later on.

This chapter uses the data set SALES to illustrate the concepts. This data set is available on the author pages for this book on the SAS Publishing site, http://support.sas.com/authors.

Overview

So, what is the Output Delivery System? Before SAS 8, procedures and DATA steps produced output that went straight to the listing window (or an LST file or the SYSOUT DD file). With ODS, the procedures and DATA steps simply produce raw data, and ODS supplies the structure and format for the output. For example, if you run a simple PROC CORR to get the correlation coefficient for two variables, ODS could create the HTML output shown on the left in Figure 2.1.

The CORR procedure did not create the tables, formatting, or even the labels for the rows and columns. The CORR procedure simply produced the raw data for the table cells. Figure 2.2 shows this raw data. (Note that Figure 2.2 is a simplification. The data is actually stored as double-precision floating-point values.) The data includes the names of the two variables, the number of observations, the Pearson correlation coefficient, and the probability result.

ODS takes this information and combines it with several *table definitions* to produce the results that are displayed in Figure 2.3. The table definitions for each procedure are supplied by the procedure. PROC CORR uses two table definitions to control the structure of its output. The first table definition tells ODS how to display the variables information table at the top of the output. It specifies a table with a column for the variable names, and it also specifies the labels for the table, row, and column headings.

A second table definition tells ODS how to display the correlation results. It specifies the column for each VAR variable, and the row to show each WITH variable. It also specifies that the table cell(s) display the correlation coefficient and probability. If this PROC CORR had been specified with additional options, such as a request for simple statistics, ODS would supply additional table definitions to structure the other types of output.

However, these table definitions are not all that ODS does. If ODS stopped here, your output would look much like the basic listing output displayed in Figure 2.3. Table definitions control the basic structure of the output and some aspects of the formatting of the output. When you create HTML, RTF, or Printer output, you also need a style definition.

The *style definitions* specify the background color; table borders and color scheme; and the fonts, sizes, and color of the text and numbers. In this case, the style definition for HTML calls for a table with standard HTML borders, which is shown on the left in Figure 2.1. With one simple change, you can generate the output on the right with the same numbers, but a different color scheme.

Finally, ODS allows you to specify *output destinations*, which determine where and how the results are stored. These destinations include HTML, as in this example, and also RTF files, PostScript and PCL printer files, PDF files, and output data sets. SAS is adding new output destinations all the time: XML, CSV, LaTeX, troff, and Excel Spreadsheet Markup Language are just a few of the output destinations available to you.

These table definitions, style definitions, and output destinations enable ODS to take simple procedure results—two variable names and two statistics generated by PROC CORR—and create the HTML output shown in Figure 2.1. ODS is an interface between your simple results and the varied and complex output you want to create.

Here is the PROC CORR step that creates the output for this chapter:

```
proc corr data=sales nosimple;
  var ItemsSold;
  with ItemPrice;
run;
```

The output on the left side in Figure 2.1 shows the PROC CORR report using the default style for HTML output. The output on the right shows the same output using the Sasweb style that SAS supplies.

Figure 2.1

PROC CORR output, in default style

The CORR Procedure

1 With Variables:	ItemPrice
1 Variables:	ItemsSold

Pearson Correlation Coefficients, N = 500 Prob > |r| under H0: Rho=0	
	ItemsSold
ItemPrice Item Price	0.04968 0.2675

PROC CORR output, in SASWEB style

The CORR Procedure

1 With Variables:	ItemPrice
1 Variables:	ItemsSold

Pearson Correlation Coefficients, N = 500 Prob > |r| under H0: Rho=0	
	ItemsSold
ItemPrice Item Price	0.04968 0.2675

Figure 2.2

ItemPrice	ItemsSold	500	0.04968	0.2675

Figure 2.3

1 With Variables:	ItemPrice
1 Variables:	ItemsSold
Pearson Correlation Coefficient, N=500	
Prob > |r| under H0: Rho=0	
	ItemsSold
ItemPrice	0.04968
ItemPrice	0.2667

Table Definitions and Style Definitions

As explained in the previous example, table definitions are used by ODS to control how procedure and DATA step results are displayed. They define the structure and basic layout for your results. What makes ODS so powerful is that you can modify these table definitions to change the structure or content of your output.

For example, our PROC CORR from the previous example creates two pieces of output: a table with the variable names, and a table with the number of observations, correlation coefficient, and probability. However, we're not limited to this output. ODS allows us to specify which tables we'd like to create.

PROC CORR has some built-in control over output. In this example, the NOSIMPLE option has already been used to get rid of one unwanted output table, the display of simple summary statistics. However, with only two variables, the table of variable names is also unwanted. The output is perfectly clear with just the correlation matrix. But the table of variable names cannot be removed from within PROC CORR.

Instead, ODS can be used to get rid of the table of variable names. Figure 2.4 shows the result. Now only the correlation matrix itself is displayed. ODS allows even further customization beyond that provided by the procedures themselves. Chapter 8, "Limiting ODS Output," covers how to select and exclude the individual tables and graphs, or output objects, that make up procedure output, thus enabling you to create customized results for each output destination.

In addition to letting us specify which output objects we want, ODS also enables us to modify the way that they are displayed. If you want to share these results with a non-programmer, the label "The CORR Procedure" might not be very helpful. You can use ODS to remove this label. You can also change or remove the table borders. Figure 2.5 shows the resulting output after the procedure label and table borders have been removed. These techniques are covered in Chapter 16, "Modifying Output Structure."

Finally, ODS allows you to change the appearance of procedure output. This is done by modifying the style definition. Style definitions control colors for backgrounds and fonts, typeface selection, and other aesthetic aspects of ODS output.

Going back to our PROC CORR example, the style definition can be modified to use a different font. Figure 2.6 shows what the output looks like if all of the fonts are changed to "Comic Sans MS." Though this output looks rather silly, it does illustrate the power of style definitions. Techniques for modifying style definitions are covered in Chapter 13, "Style Template Definitions."

In addition to table template definitions and style template definitions, two other major template types are used with ODS. They are tagset template definitions, used for ODS MARKUP destinations, and graph template definitions, used by ODS Graphics. Tagset

templates are discussed in Chapter 12, “Using Tagsets to Generate Markup Output.” Graph templates are discussed briefly in Chapter 20, “Understanding ODS Graphics Output.”

Figure 2.4

The CORR Procedure

Pearson Correlation Coefficients, N = 500 Prob > |r| under H0: Rho=0	
	ItemsSold
ItemPrice Item Price	0.04968 0.2675

Figure 2.5

Pearson Correlation Coefficients, N = 500 Prob > |r| under H0: Rho=0	
	ItemsSold
ItemPrice Item Price	0.04968 0.2675

Figure 2.6

Pearson Correlation Coefficients, N = 500 Prob > |r| under H0: Rho=0	
	ItemsSold
ItemPrice Item Price	0.04968 0.2675

Output Objects and Output Destinations

After the table definitions are applied, each SAS procedure can then produce output objects. In the PROC CORR example, two output objects were created: the list of variables and the correlation matrix. These output objects can then be sent to one or more output destinations.

The default output destination is traditional SAS listing output.

On the facing page, Figure 2.7 through Figure 2.10 show a simple result that has been routed to a number of destinations, creating a number of pieces of ODS output. The PROC CORR step was run only once, producing the raw data that is combined with the table definition to create the basic output object. Then the style definitions interact with the output objects to create the output for each destination.

This is a simple example with just a single output object. Many procedures by default produce a number of output objects.

Figure 2.7 shows the LISTING destination. Even though the table definition was applied to configure the output, the style definition does not apply to the LISTING destination.

Figure 2.8 shows the HTML destination. Here many aspects of the style definition apply, so we get colors and an assortment of typefaces and font sizes.

Figure 2.9 shows the RTF destination. This output looks very similar to the HTML output. There are just minor differences in how the style definitions are implemented to create RTF output.

Figure 2.10 shows the PDF destination. This output looks very similar to the RTF and HTML output. The look and feel of the reports from these three destinations can vary somewhat, depending on the style definition that you use to create the output files.

Finally, Figure 2.11 shows the OUTPUT destination, which creates a data set. Here only the table definition comes into play, and it determines the structure of the raw data. Style definitions are not used since the output is a data set.

In this example, one procedure that turns out one result can be routed to five different output destinations, creating five unique pieces of ODS output.

Other possible output destinations include LaTeX, CSV, and Excel.

To help you keep track of all of these output objects, SAS provides the Results window. If you look at Figure 2.12, you can see a view of the Results window for the output shown in these examples. This window displays each of the output objects you have created, along with an icon to indicate the output destinations to which it has been sent.

Figure 2.7: Listing Output

```
Pearson Correlation Coefficients, N
= 500
Prob > |r| under H0: Rho=0

                    Items
                     Sold

Item Price        0.04968
                   0.2675
```

Figure 2.8: HTML Output

Pearson Correlation Coefficients, N = 500 Prob > |r| under H0: Rho=0	
	ItemsSold
ItemPrice Item Price	0.04968 0.2675

Figure 2.9: RTF Output

Pearson Correlation Coefficients, N = 500 Prob > |r| under H0: Rho=0	
	ItemsSold
ItemPrice Item Price	0.04968 0.2675

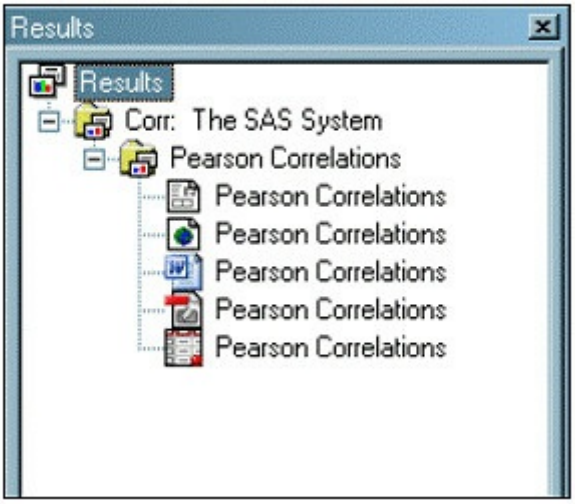
Figure 2.10: PDF Output

Pearson Correlation Coefficients, N = 500 Prob > |r| under H0: Rho=0	
	ItemsSold
ItemPrice Item Price	0.04968 0.2675

Figure 2.11: Output data set

	Variable	Label	ItemsSold	PItemsSold
1	ItemPrice	Item Price	0.04968	0.2675

Figure 2.12



ODS References and Resources

There is no better set of reference guides to ODS than the new tip sheets that were developed by the Base SAS Team, as shown on the Base SAS R&D Web site. The tip sheets should not be considered a substitute for the ODS documentation. However, they are concise, accurate, and extremely handy. And because they are maintained by the ODS developers, they represent the most up-to-date quick reference sheets available.

For your convenience, the URLs to download each of the ODS tip sheets (in PDF form) are listed on the author pages for this book: http://support.sas.com/authors. Tip sheets available at the time of printing include the following:

- ODS Basics and ODS HTML
- ODS PDF
- ODS RTF
- Using ODS Style Syntax with PROC PRINT, REPORT and TABULATE
- Style Template Syntax
- Table Template Syntax
- ODS DOCUMENT
- ODS TAGSETS.RTF
  (new in SAS 9.2)

In addition to these handy tip sheets, on the R&D Web site you will find links to papers by the developers and the most current information about R&D efforts in the Base SAS arena.

Part 2
The Basics

Chapter 3 HTML Output 23
Chapter 4 RTF Output 81
Chapter 5 PDF Output 151

Chapter 3
HTML Output

This chapter explains how to generate HTML output and it provides a basic description of the HTML that ODS generates. It shows a few of the enhancements you can add to take your results to the next level. Finally, it shows how to open your HTML results with a word processor, spreadsheet, or other package for further customizing.

HTML output generated by ODS uses enhanced fonts, colors, and table designs that make the most of your results. It can also organize your output by creating a table of contents, and it can incorporate your output into an existing Web site.

SAS®9 included numerous enhancements to the HTML destination. With SAS®9, you can create many different kinds of HTML markup files, including but not limited to HTML 3.2-compliant tags, HTML 4.0-compliant tags, and Microsoft Office HTML tags. You can also create XHTML and Microsoft Excel XML tags.

This chapter uses three data sets, BILLINGS, CLINICAL, and TOTALS, to illustrate how to produce HTML output with ODS. These data sets are available on the author pages for this book at http://support.sas.com/authors.

Generating Your First HTML File

Creating an HTML file from your SAS results is quite simple. You need just three things: a file to send the results to, an instruction to ODS to open the file and start generating HTML tags, and an instruction to ODS to stop generating HTML tags and close the file.

The first thing you need to do is specify a file for your output. You can specify the file directly when you call ODS to create the HTML output, or you can create a filename reference ahead of time and use that in the ODS call. If you choose to create a filename reference, the syntax, which is coded specific to your operating system, is something like this:

```
filename myfile 'myhtmlfile.html';
```

The filename should end in ".html" (or ".htm") in order for your operating system to choose a browser to open and render your file. This example is for SAS under Windows. The file naming conventions vary by operating system, and you might need to make adjustments (see the Appendix, "Operating System Differences," for more information). After you've created the filename, you can then issue your call to ODS to request HTML output. Here is the syntax:

```
ODS HTML BODY=myfile;
```

If you did not choose to create a fileref, the syntax would change to this:

```
ODS HTML BODY='myhtmlfile.html';
```

This is all it takes to turn on ODS HTML output. The output from every procedure or DATA step following this statement is directed to the HTML file you specify. This is in addition to the standard listing output, which is directed to the output window as always. After you have run the procedures you want to send to the HTML file, you need to turn the HTML destination off so that SAS closes the file. Then you can view your HTML file with the SAS Results Viewer (on Windows) or your favorite browser. Here is the syntax to end HTML generation:

```
ODS HTML CLOSE;
```

That's the last of the three pieces you need to create HTML output. Viewed all together, they look like this:

```
filename myfile 'myhtmlfile.html';
ODS HTML BODY=myfile;
proc means data=billings maxdec=2;
  class quarter;
  var Hours BillableAmt;
run;
ODS HTML CLOSE;
```

In this example, the output from PROC MEANS is stored in the file referenced by myfile.

The HTML output, viewed in a browser or the SAS Results Viewer, is shown in Figure 3.1.

Figure 3.1

The MEANS Procedure

Quarter	N Obs	Variable	Label	N	Mean	Std Dev	Minimum	Maximum
2009-1	62	Hours	Hours Worked	62	5.98	2.95	1.00	11.00
		BillableAmt	Amount Billable	62	571.79	265.04	119.79	1022.80
2009-2	60	Hours	Hours Worked	60	6.48	2.86	1.00	11.00
		BillableAmt	Amount Billable	60	617.60	257.19	118.80	1030.80
2009-3	64	Hours	Hours Worked	64	6.25	2.79	1.50	11.00
		BillableAmt	Amount Billable	64	596.38	251.04	165.81	1028.81
2009-4	66	Hours	Hours Worked	66	5.86	2.70	1.00	11.00
		BillableAmt	Amount Billable	66	559.99	243.30	115.82	1024.82
2010-1	52	Hours	Hours Worked	52	6.08	2.80	1.00	10.50
		BillableAmt	Amount Billable	52	580.97	250.74	126.83	984.83
2010-2	63	Hours	Hours Worked	63	6.02	2.70	1.00	10.50
		BillableAmt	Amount Billable	63	574.82	243.35	119.84	984.84
2010-3	69	Hours	Hours Worked	69	5.83	2.97	1.00	11.00
		BillableAmt	Amount Billable	69	557.76	265.92	121.85	1025.85
2010-4	64	Hours	Hours Worked	64	6.02	2.81	1.00	11.00
		BillableAmt	Amount Billable	64	574.36	252.54	119.86	1017.86

Exploring the Body File: The Basics of HTML

When ODS creates HTML output, it takes the results that would normally go only to the SAS Output window (or LISTING destination) and renders them as an HTML file that can be understood by Web browsers like Internet Explorer or Firefox. The SAS LISTING destination results are just text and numbers separated by spaces and line breaks when displayed in the Output window. For the results to be displayed in a Web browser, they need to be coded with HTML *tags*. These tags control things like the display layout in the browser window and the fonts and colors used for the text and numbers.

To illustrate this process, we'll look at a simple bit of ODS HTML output. The following code creates an HTML file called procmeans.html that contains a table with summary statistics for a single variable. The resulting HTML output is shown in Figure 3.2.

```
      ODS HTML BODY='procmeans.html';
proc means data=billings nonobs maxdec=2 mean min max;
   var BillableAmt;
run;
ODS HTML CLOSE;
```

To see how ODS has created this Web page, look at Figure 3.3, which shows what the HTML code looks like. ODS generates more HTML than is actually shown in Figure 3.3. However, for the purposes of understanding the basic structure of the HTML output, only the essential HTML code is discussed and displayed.

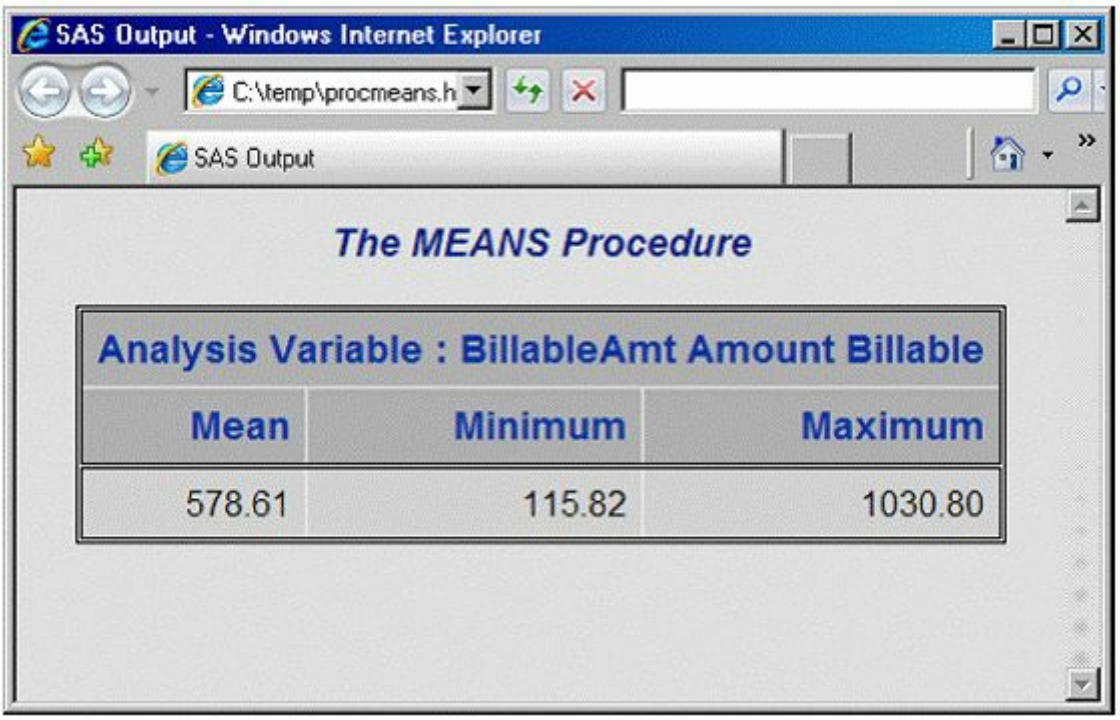
What makes this an HTML file are the <HTML> and </HTML> tags at the beginning and end of the file. The <HEAD> and </HEAD> tags coupled with the <TITLE> and </TITLE> tags enclose the text ("SAS Output") that is displayed in the browser title bar. Everything between <BODY> and </BODY> is displayed in the main window of the browser.

Browsers are not very good at lining up text unless you are using fixed-width fonts, so to align the output for the browser, all of the results are organized into tables with rows and columns. This output has one table for the actual results. The <TABLE> and </TABLE> tags mark the beginning and end of the table. Within the table, <TR> and </TR> mark the beginning and end of each row. Finally, within each row, <TD> and </TD> mark the beginning and end of each table cell. The table has three rows. The first row contains the table heading, and has only one cell. The second row has the column headings, and has three cells. The third row contains the numeric results, and also has three cells.

The example HTML file also includes an inline style section, which determines things such as font face, font size, background color, etc. This is a very lengthy section that is not shown here. The inline style section is linked to the table cells by the use of CLASS= attributes, which you can see in Figure 3.3. The actual results displayed in Figure 3.2 are generated by ODS, which produces HTML
4.01-compliant tags, creating an HTML file that should open in most browsers.

Although this example is an oversimplification, it does provide an idea of how the basics of HTML files work, and how they are used to display your SAS output.

Figure 3.2



If you don't like the generic "SAS Output" title that appears in the browser title bar, you can change the text using the TITLE= suboption in your ODS HTML statement. Note that the TITLE= suboption belongs immediately after the BODY= or FILE= option.

```
ODS HTML BODY='procmeans.html' (TITLE='My Title');
```

Figure 3.3

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<body onload="startup()" onunload="shutdown()" class="Body">
<div class="1 ProcTitle">The MEANS Procedure</div>
<table class="Table" cellspacing="1" cellpadding="7" rules="groups" frame="box"
border="1px" bordercolor="#000000" summary="Procedure Means: Summary statistics">
<thead>
<tr><th class="c b Header" colspan="3" scope="colgroup">Analysis Variable :
BillableAmt Amount Billable</th></tr>
<tr>
<th class="r b Header" scope="col">Mean</th>
<th class="r b Header" scope="col">Minimum</th>
<th class="r b Header" scope="col">Maximum</th>
</tr></thead>
<tbody><tr>
<td class="r Data">578.61</td>
<td class="r Data">115.82</td>
<td class="r Data">1030.80</td>
</tr></tbody></table>
</body>
</html>
```

Exploring the HTML Body File: Graphics

Most ODS HTML output is created as shown in the previous example. The results are organized into tables with rows and columns of headings and statistics. However, graphics procedures are different. They produce an image, rather than text results surrounded by HTML tags. Graphics output is covered in detail in Chapter 20, “Understanding ODS Graphics Output,” but this simple example will give you a basic understanding of graphical output contained in an HTML result file.

The ODS statements you use to create HTML graphics output are nearly identical to those used for creating tabular output. You still need the three basics: an ODS HTML statement to open the HTML output destination, a body filename to specify where to send the output, and an ODS HTML CLOSE statement to close the HTML output destination.

For graphics procedures, you need three extra things. First, you need to specify the device driver, and SAS/GRAPH has a lot of them. Some popular ones are GIF, JPEG, PNG, or ActiveX. The GIF, PNG, or JPG formats are easily interpreted by Web browsers. The second thing you might use is a PATH or GPATH option on your ODS HTML statement. These options tell ODS where to put the HTML and GIF files when created. Generally, you want to point the GPATH to the same location as your body file (PATH) location. Third, you should add URL=NONE. This keeps the graphics image physical storage path from being included in the HTML output, which allows you to move the resulting HTML and GIF output to your Web server.

```
      GOPTIONS DEVICE=GIF;
ODS HTML PATH='c:\temp' (URL=NONE) GPATH='c:\temp' (URL=NONE)
       BODY='c:\temp\mygraph.html';
proc gchart data=Billings;
  vbar3d Quarter /
    name="billqtr"
    sumvar=BillableAmt type=mean;
run;
      quit;
ODS HTML CLOSE;
```

Submitting this code in SAS 9.2 generates Figure 3.4. This HTML page contains a single image with the chart and all of the associated text. A highly simplified selection of the HTML code generated by ODS is shown in Figure 3.5. The only part of the HTML code shown is the <IMG> tag that SAS generates. This tag points to the GIF file. Instead of text results and labels, this example shows that SAS can generate HTML code that points to images.

Most images are displayed in browsers by using the <IMG> tag, whether they’re your corporate logo or an image created with SAS/GRAPH. Unlike the other tags in this example and the previous examples, this tag is not one of a pair: there is no </IMG> tag. Also, the <IMG> tag uses parameters. The value for the ALT= parameter comes from the NAME= option in the VBAR statement. The SRC= parameter specifies the name of the image file: src=“billqtr.gif”. The third parameter, STYLE=, specifies formatting characteristics of the

output; this example specifies that no border be displayed around the graphic image.

This HTML output was generated using the default settings for the graph. One important difference between the graphical output created in SAS 9.2 compared to earlier releases of SAS is that SAS style templates are used in the creation of SAS/GRAPH output in SAS 9.2. The colors that you see in the chart are the default graph colors defined in the STYLES.DEFAULT style template definition.

Figure 3.4

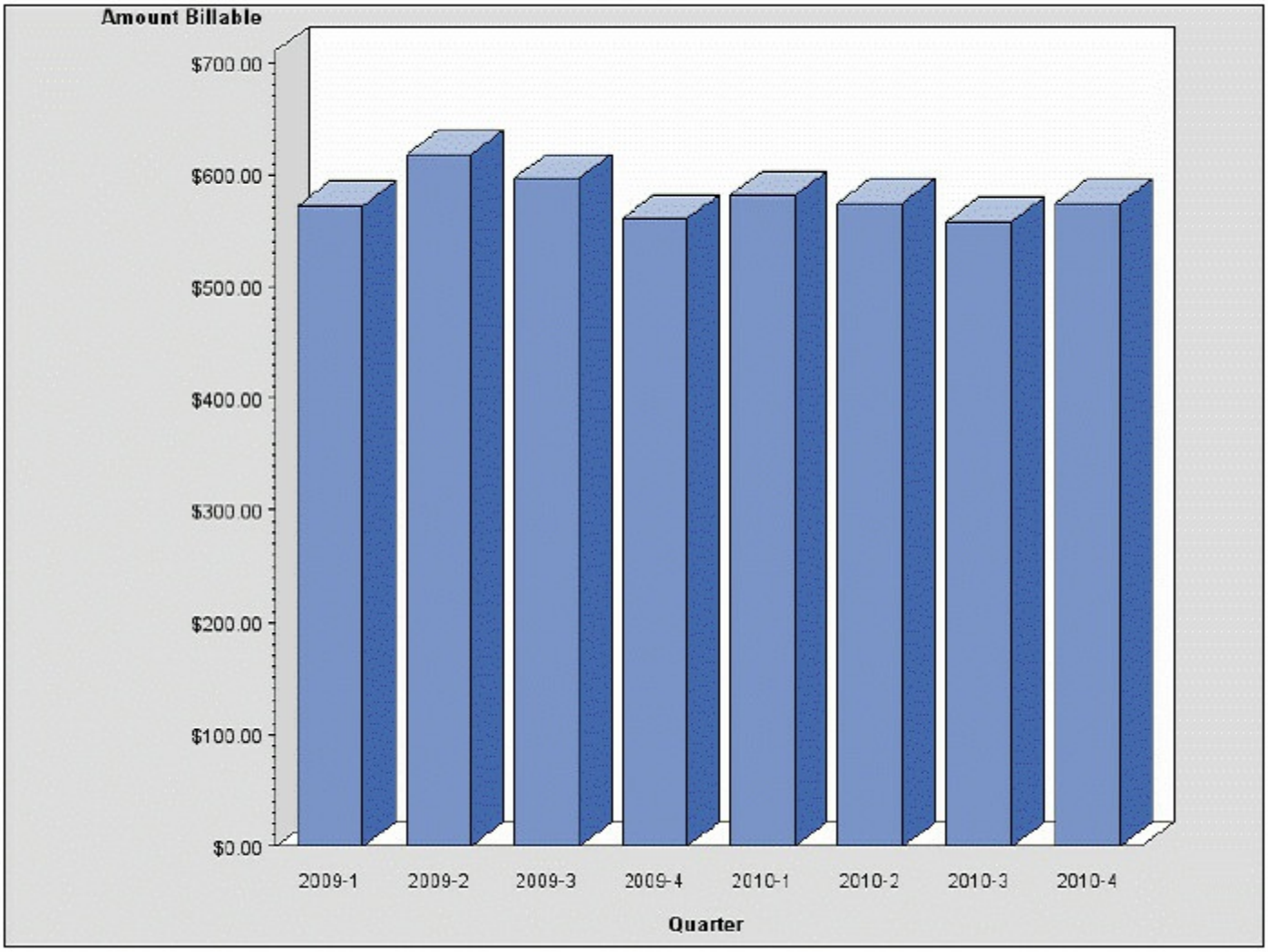


Figure 3.5

```
<html>
<div class="branch">
<a name="IDX"></a>
<div><div  class="c">
<img alt="Bar chart of Quarter" src="billqtr.gif" style=" border-width: 0px;
height: 600px; width: 800px;" border="0" class="c Graph">
</div></div>
<br>
</div>
</body>
</html>
```

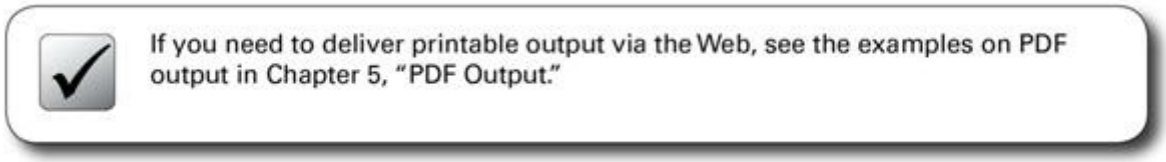
Combining the Results of Several Procedures

The previous two examples showed how to send output from a single procedure to an HTML file. However, ODS also enables you to send a number of results from a series of procedures to the same HTML file. This allows you to build a combined report from a number of procedures and data sets and place it all on a single Web page. In this example, the output from three procedures is combined into a single HTML file. Two PROC MEANS steps and a PROC PRINT step are all placed between two ODS HTML statements.

```
      title 'Quarterly Report';
ODS HTML BODY='QReport.html';
title2 'Revenue by Customer';
proc means data=Billings nonobs mean sum;
  class CustomerName;
  var BillableAmt;
run;
title2 'Revenue by Division';
proc means data=Billings nonobs mean sum;
   class Division;
   var BillableAmt;
run;
title2 'Top Performers';
proc print data=Totals noobs;
   where BillableAmt>10000;
   var EmployeeName BillableAmt;
run;
ODS HTML CLOSE;
```

The data set TOTALS was created from the BILLINGS data set in a separate step (not shown here).

Because all output between the ODS HTML and ODS HTML CLOSE statements is routed to the HTML destination, the output from all three procedures goes to the file specified in the BODY= option. The resulting file is shown in Figure 3.6. In this screen view, you can see the results from the first PROC MEANS step and the top of the results from the second PROC MEANS step. To see more, you would have to scroll down in the browser window.



If you need to deliver printable output via the Web, see the examples on PDF output in Chapter 5, "PDF Output."

SAS builds this HTML file by appending the results of each procedure to the end of the HTML file. If you run a large number of procedures, you'll end up with an extremely long file that requires a lot of scrolling to see all the results. (The NEWFILE= option can help with this. See "Breaking Up Output into Separate Files.") A horizontal bar is added between each type of output, but you will notice that there are no page numbers; and if you print this HTML file, the browser might or might not put a page break at the horizontal rule. That is because even though SAS puts a CSS instruction into the HTML file, *HTML does not have any code or tag to mark a page break*. HTML is not a great printing tool. When you print an HTML file, it is

the browser, and not SAS, that controls the printing.

Figure 3.6

Quarterly Report
Revenue by Customer

The MEANS Procedure

Analysis Variable : BillableAmt Amount Billable		
Customer	Mean	Sum
Bricks and Mortar, Inc.	614.6238202	54701.52
IPO.com	612.9017204	56999.86
Smith & Smith	547.2538182	60197.92
Sweat Shops Athletic	581.4485185	62796.44
Virtual Co.	546.0762000	54607.62

Quarterly Report
Revenue by Division

The MEANS Procedure

Analysis Variable : BillableAmt Amount Billable		
Division	Mean	Sum
Analysis	541.1286585	44372.55
Applications	565.2143333	50869.29
Documentation	587.1271250	46970.17

Generating a Table of Contents

The previous example showed how the results of several procedures could be combined into a single HTML output file. This makes a nice summary report, but it does require the reader to scroll through several pages of output. To make this report even better, you can use ODS to generate a table of contents that allows the reader to click on an item of interest and jump right to that item.

The code involved is quite simple and requires only the addition of two more options to the ODS HTML statement. When you request a single HTML file for your output, you use a BODY= option. To get a table of contents, you add a CONTENTS= option. You should also add a FRAME= option. The CONTENTS= option creates the table of contents, but in order for a Web browser to display both the table of contents and the body file in the same browser window, you need a frame file, which is specified with the FRAME= option.

```
      title 'Quarterly Report';
ODS HTML BODY='body.html'
       CONTENTS=’contents.html’
       FRAME='frame.html';
title2 'Revenue by Customer';
proc means data=Billings nonobs mean sum;
   class CustomerName;
       var BillableAmt;
      run;
      title2 'Revenue by Division';
      proc means data=Billings nonobs mean sum;
          class Division;
          var BillableAmt;
      run;
      title2 'Top Performers';
      proc print data=Totals noobs;
        where BillableAmt>10000;
        var EmployeeName BillableAmt;
      run;
      ODS HTML CLOSE;
```

In this example, the body file is called body.html, the contents file is called contents.html, and the frame file is called frame.html. To view the results, you open the frame file. As you can see in Figure 3.7, the frame file organizes the body and contents files. The contents file is displayed in the left frame, and the body file is displayed in the right frame. If you were to click on the label for a procedure, the browser would take you directly to that output, with no scrolling required.

This kind of frame navigation uses standard HTML tags to link the files together for display in one browser window. Your company webmaster might use JavaScript, Java, ActiveX, or PHP to generate more dynamic HTML navigation. It’s a good idea to verify whether your webmaster allows standard HTML frames on the company Web site.

Figure 3.7

Table of Contents

1. The Means Procedure
   - Summary statistics
2. The Means Procedure
   - Summary statistics
3. The Print Procedure
   - Data Set WORK.TOTALS

Quarterly Report
Revenue by Customer

The MEANS Procedure

Analysis Variable : BillableAmt Amount Billable		
Customer	**Mean**	**Sum**
Bricks and Mortar, Inc.	614.6238202	54701.52
IPO.com	612.9017204	56999.86
Smith & Smith	547.2538182	60197.92
Sweat Shops Athletic	581.4485185	62796.44
Virtual Co.	546.0762000	54607.62

In this example, the HTML output files were named to match their function. The contents file is named contents.html, the body file body.html, and the frame frame.html. This is handy because it helps you remember what each file does, but it is not required. You can call the files anything you want (x.html, y.html, and z.html, for example) and SAS will still build the proper structure and cross links.

As you look at this output, you might be thinking that it would be nice if the table of contents displayed the title instead of the procedure label, "The Means Procedure." In fact, you can do that. See the example on the PROCLABEL option later in this chapter.

Exploring the Frame File

In the previous example, ODS was used to generate a table of contents to organize the results. To set up this output, three files were created. The first file was the body file, which contained the actual results. The second file was the contents file. This contains the procedure names and links to each procedure's output. The HTML code for this file is shown in Figure 3.8. This code has been greatly simplified. The actual file contains several complex scripts that enable you to expand or collapse the table of contents items in the browser. Nevertheless, the important part of the code is shown here.

This file is composed of a series of titles: "The MEANS Procedure," "The PRINT Procedure," etc. Each title is followed by a subtitle that further explains the output: "Summary Statistics," "Data Set WORK.TOTALS." The subtitles are links to that part of the output in the body file. Notice the
<A HREF= TARGET=> and </A> tags. The <A> and </A> mark the beginning and end of a hyperlink. The text between them is the text that is displayed in the table of contents. The HREF= attribute specifies the location to link to (in this case, various locations in the body file). Finally, the TARGET= attribute tells the browser to continue to display the body file in the main frame of the window. These links and references are further explained in an example later in this chapter.

The third file involved in displaying the output with a table of contents is the frame file. This tells the browser how to display the body file and contents file. The code for the frame file is shown in Figure 3.9. This file is very simple. Instead of having a body section (note that there are no <BODY> tags), this file has <FRAMESET> and </FRAMESET> tags. These tags have a number of attributes, but the important one is the COLS attribute. This tells the browser how many columns the page will have and how big they will be. In this case, there are two columns. The first one takes up 23% of the page width, and the second one gets the remainder of the page.

Inside the <FRAMESET> tags are two sets of <FRAME> and </FRAME> tags. These specify which files will be displayed in the two columns on the screen. They indicate that the left frame will be used to display the file contents.html, and the right frame will be used to display body.html.

Together, these three files comprise a frame Web page. SAS and ODS set up all of these files for you automatically, but it's useful to know how they work.

If you are incorporating your SAS output into an existing Web site, be careful about how you set up your pages. If your Web site is already set up with frames, and you add a link to your SAS output, which has its own frames, you could end up with ugly and confusing output. This "frames-within-frames" problem is illustrated in Figure 3.10. To solve this problem, code the hyperlink on your existing Web page to open up your ODS output in a new window. You do this by adding the code TARGET="_blank" to your hyperlink in your existing Web page (not the ODS-generated page).

For additional information about how the table of contents page works, see "Exploring Links and Anchors."

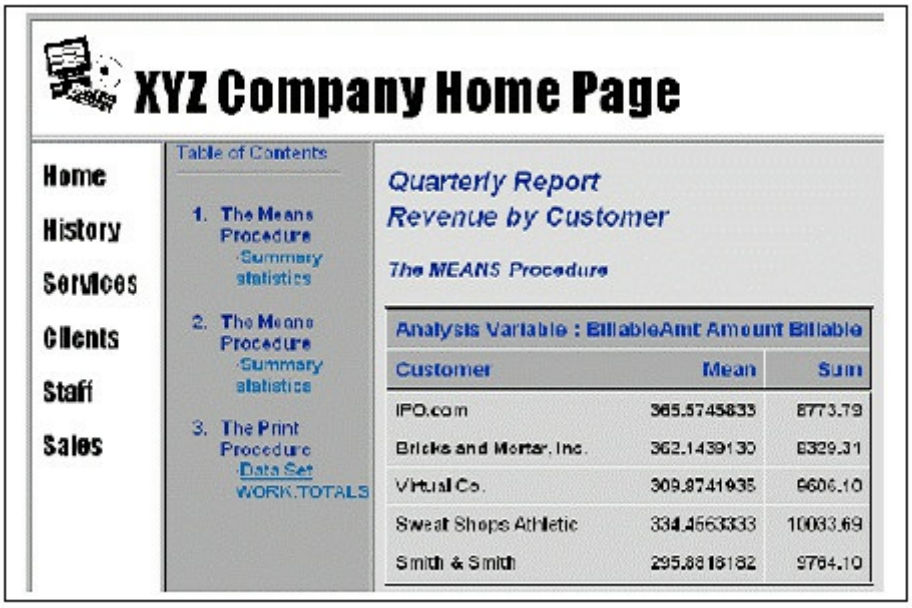
Figure 3.8

```
<HTML>
   <HEAD>
      <TITLE>SAS Output</TITLE>
   </HEAD>
   <BODY>
   Table of Contents<br>
   The Means Procedure<br>
      <A HREF="body.html#IDX" TARGET="body">Summary statistics</a><br>
   The Means Procedure<br>
      <A HREF="body.html#IDX1" TARGET="body">Summary statistics</a><br>
   The Print Procedure<br>
      <A HREF="body.html#IDX2" TARGET="body">Data Set WORK.TOTALS</a><br>
   </BODY>
</HTML>
```

Figure 3.9

```
<HTML>
   <HEAD>
      <TITLE>SAS Output</TITLE>
   </HEAD>
   <FRAMESET cols="23%,*">
      <FRAME MARGINWIDTH="4" MARGINHEIGHT="0" FRAMEBORDER="0"
            SRC="C:\TEMP\CONTENTS.HTML" NAME="CONTENTS" SCROLLING="AUTO"
            TITLE="THE TABLE OF CONTENTS">
     <FRAME MARGINWIDTH="9" MARGINHEIGHT="0" FRAMEBORDER="0"
           SRC="C:\TEMP\BODY.HTML" NAME="BODY" SCROLLING="AUTO"
           TITLE="THE SAS OUTPUT">
   </FRAMESET>
</HTML>
```

Figure 3.10


XYZ Company Home Page
Home
History
Services
Clients
Staff
Sales
Table of Contents
1. The Means Procedure
Summary statistics
2. The Means Procedure
Summary statistics
3. The Print Procedure
Data Set WORK.TOTALS
Quarterly Report
Revenue by Customer
The MEANS Procedure
Analysis Variable : BillableAmt Amount Billable
Customer Mean Sum
IPO.com 365.5745833 8773.79
Bricks and Mortar, Inc. 362.1439130 8329.31
Virtual Co. 309.8741935 9606.10
Sweat Shops Athletic 334.4563333 10033.69
Smith & Smith 295.8818182 9764.10

Breaking Up Output into Separate Files

All of the examples thus far have shown the results in a single body file. To view each of the results, you have to scroll down the page or click on the appropriate table of contents link. If you want to print one of the results, you have to print the whole page. If your output is lengthy, or your users need to access it via a slow connection, you might want to break your output up into smaller files.

The ODS HTML statement supports an option that creates a separate body file for each procedure. Alternatively, you can put the results from a few procedures in one body file, and the results from the next few procedures in another. The following code shows how to take our sample program from the previous examples and write it to three body files, one for each procedure.

```
      ODS HTML BODY='body1.html' NEWFILE=PROC
       CONTENTS='contents.html'
       FRAME='main.html';
title2 'Revenue by Customer';
proc means data=Billings nonobs mean sum;
   class CustomerName;
  var BillableAmt;
run;
title2 'Revenue by Division';
proc means data=Billings nonobs mean sum;
   class Division;
   var BillableAmt;
run;
title2 'Top Performers';
proc print data=Totals noobs;
   where BillableAmt>10000;
   var EmployeeName BillableAmt;
run;
ODS HTML CLOSE;
```

ODS also supports the creation of a table of pages, as an alternative to a table of contents (or you can have both a table of contents and a table of pages). To add a table of pages, add PAGE='fileref' to your ODS HTML statement. Figure 3.12 shows how this example would look with a table of pages instead of a table of contents.

In this code, there are just two changes. First, the name of the body file is changed to body1.html. The second change is the addition of the NEWFILE=PROC option, which calls for a new HTML file for each procedure. The first file is named body1.html, the next file is named body2.html, and so on. The contents file will include links to the multiple files. The results are shown in Figure 3.11. You can also start a new file for each page (NEWFILE=PAGE), each output table (NEWFILE=OUTPUT), or each BY group (NEWFILE=BYGROUP). The results look just like the previous output, but now if you print the screen, you'll get only the output from a single procedure. And if you scroll down in this output, you cannot get to the next procedure. You have to use the table of contents to get to the next procedure.

Figure 3.11

Table of Contents

1. The Means Procedure
   ·Summary statistics
2. The Means Procedure
   ·Summary statistics
3. The Print Procedure
   ·Data Set WORK.TOTALS

Quarterly Report
Revenue by Customer

The MEANS Procedure

Analysis Variable : BillableAmt Amount Billable		
Customer	**Mean**	**Sum**
Bricks and Mortar, Inc.	614.6238202	54701.52
IPO.com	612.9017204	56999.86
Smith & Smith	547.2538182	60197.92
Sweat Shops Athletic	581.4485185	62796.44
Virtual Co.	546.0762000	54607.62

Figure 3.12

Table of Pages

1. The Means Procedure
   ·Page 1
2. The Means Procedure
   ·Page 2
3. The Print Procedure
   ·Page 3

Quarterly Report
Revenue by Customer

The MEANS Procedure

Analysis Variable : BillableAmt Amount Billable		
Customer	**Mean**	**Sum**
Bricks and Mortar, Inc.	614.6238202	54701.52
IPO.com	612.9017204	56999.86
Smith & Smith	547.2538182	60197.92
Sweat Shops Athletic	581.4485185	62796.44
Virtual Co.	546.0762000	54607.62

Simulating Page Breaks Using Cascading Style Sheets

One of the first questions people ask about HTML output is how they can set it up so that the pages break properly when they print their results from a Web browser. The short answer: it depends.

A Web browser is a tool meant for looking at files, not printing them. HTML output flows continuously, so the whole concept of a page break is foreign to a browser. Some browsers support page breaks in Cascading Style Sheet (CSS) format. In that case, the default ODS style creates HTML output with proper page breaks, as shown in Figure 3.13.

The output is too small to read, but you can see that the pages break nicely between tables. This sample output contains the results from a PROC TABULATE step followed by the result of a PROC UNIVARIATE step with BY groups. Because these example pages were printed from a browser that supports CSS page breaks, the Web pages printed correctly.

However, if your Web browser does not support CSS page breaks, or you use an ODS style such as Minimal that does not include the CSS page break command, your results would print more like Figure 3.14. In this example, there are tables that span pages, which make them very hard to read when printed.

Currently, the CSS page break command works with Internet Explorer and other browsers that support this CSS style property. If you are creating Web output for your corporate intranet, and you know that all of your users will be accessing the site with a browser that supports CSS page breaks, then you might have no problem with page breaks.

That said, there are a few workarounds that might help you out. First, make sure you test your browser with a style definition that uses CSS page breaks. If you run PROC TEMPLATE to print out the style template source code, you should see something like the following code in the HTML style element.

```
'PageBreakLine' = %nrstr("<p style=""page-break-after: always;""&gt; </p><HR size=3>")
```

You'll see this text string used in the Body style element:

```
pagebreakhtml = html('PageBreakLine');
```

If you don't see these lines in your style definition, then either switch to a style that includes them, or create a new style and add them. If you've confirmed that your style is sending the page break command, but your browser or your users' browsers still don't honor the page breaks, then you'll need to try a different approach.

Figure 3.13

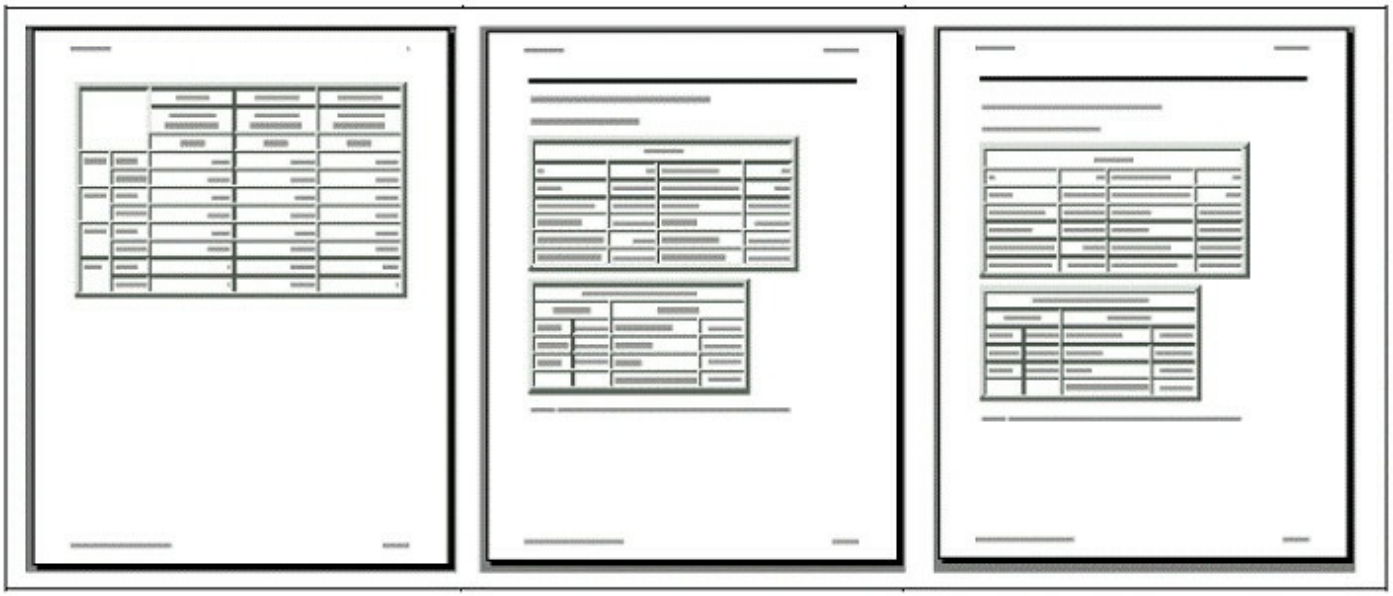
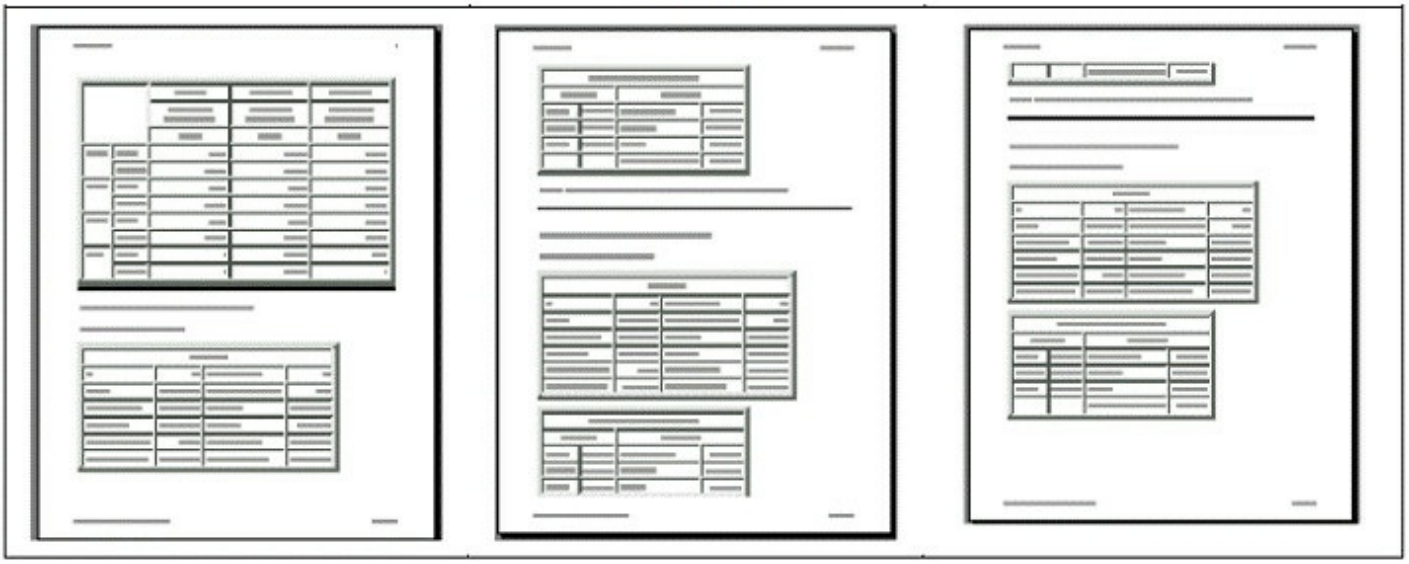


Figure 3.14



More on Page Breaks Using the NEWFILE= Option

We have already discussed using the NEWFILE= option. In comparison to the previous example, you can use the NEWFILE= option to have ODS start a new file for each procedure or for each BY group.

If you generate a table of contents, the user can easily navigate from page to page and print out a single item, which ideally will fit on one page. The catch to this is that if your output is 300 pages long, you probably don't want to have your users printing all 300 pages one page at a time.

Figure 3.15 shows the resulting pages that the user could print if you used NEWFILE=PROC in the same code as that used for the CSS example. The first page shown comes from the PROC TABULATE step. The second two pages come from the PROC UNIVARIATE step. Notice that the first page now prints nicely, but when users navigate to the second page, they get both of the BY group results, so they still have output spreading across the page.

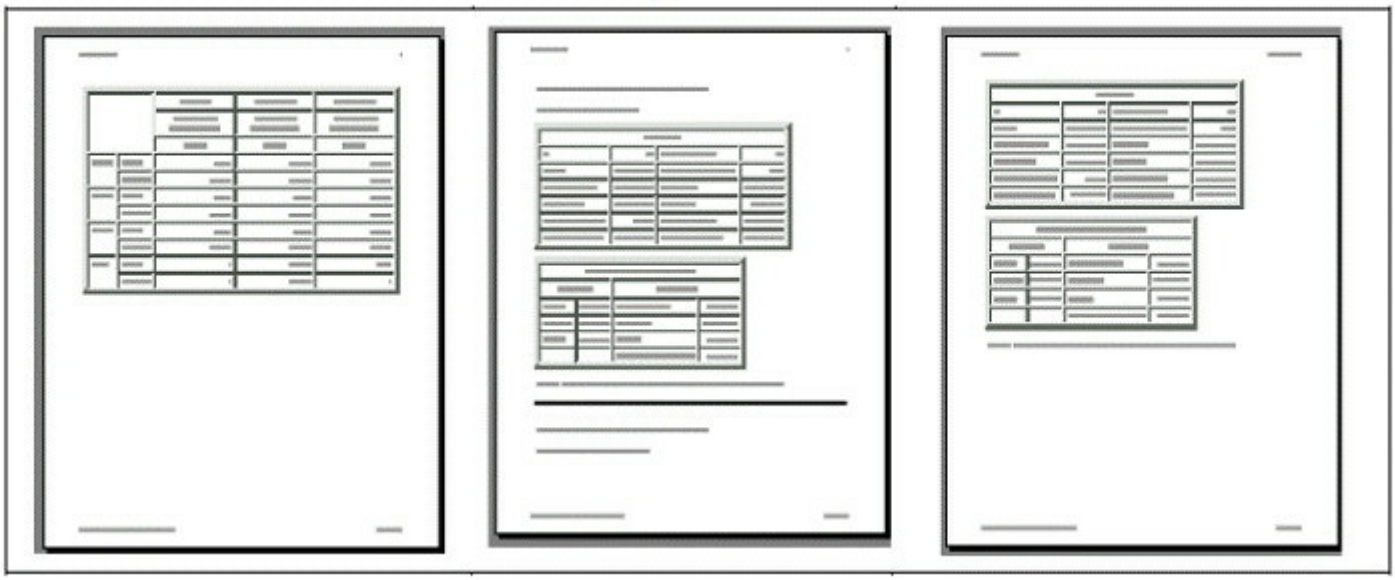
You could fix this by using NEWFILE=BYGROUP, but then you'd get the PROC TABULATE report and the report for the first PROC UNIVARIATE BY group printing together, and then the second BY group on its own page. You can't use both NEWFILE=PROC and NEWFILE=BYGROUP in the same run.

No matter which method you use to separate your output into smaller units for printing purposes, only thorough testing will ensure that you've chosen the approach that's best for you and your users.

Another way around the page break problem is to create a Web page for viewing, and add a link to the PDF version of the same output. Users can browse the Web version and print the PDF version. For example, if your PDF filename is PRTVERS.PDF, you could add a TITLE statement to the HTML file using the LINK= option in your code:

```
TITLE link='http://www.websrv.com/prtvers.pdf'
        "Click here for printable format";
```

Figure 3.15



Changing Procedure Labels in the Table of Contents

In the previous examples, you might have noticed that every time a table of contents was generated for a PROC MEANS step, the heading in the table of contents was displayed as "The MEANS Procedure," and the subheading was always "Summary Statistics." This could get very confusing if your SAS program generates fifty PROC MEANS steps. You wouldn't be able to tell them apart in the table of contents. To help with this, ODS provides a statement to specify a custom procedure label.

The ODS PROCLABEL statement is shown in the following code. In this example, the TITLE2 statement calls for the output to be headed by the title "Hours Billed by Customer." The ODS PROCLABEL statement asks for the same thing, except that it applies to the table of contents and the Results window.

```
      ODS HTML BODY='body.html' CONTENTS='contents.html'
       FRAME='main.html';
ODS PROCLABEL 'Hours Billed by Customer';
title2 'Hours Billed by Customer';
proc means data=Billings nonobs mean sum maxdec=1;
   <additional Means statements>
run;
ODS HTML CLOSE;
```

The results are shown in Figure 3.16. Notice the revised procedure label in the table of contents. It now reads "Hours Billed by Customer." The subheading "Summary Statistics" remains the same. You can also use this technique to get rid of the procedure labels in the table of contents (it has no effect on the main file). This can save space on your Web page, though you might lose some clarity. To remove the procedure labels, assign a blank label to the procedure, as in the following example:

```
      ODS HTML BODY='body.html'  CONTENTS='contents.html'
       FRAME='main.html';
ODS PROCLABEL ' ';
title2 'Hours Billed by Customer';
proc means data=Billings nonobs mean sum maxdec=1;
   <additional Means statements>
run;
ODS HTML CLOSE;
```

The ODS PROCLABEL statement also affects the PDF and RTF table of contents, and it changes the top-level output node that you see in the SAS Results window. Because ODS PROCLABEL doesn't affect your output file, consider investigating the ODS NOPTITLE option if you want to suppress the procedure title in the output file (not just the contents file).

The output is shown in Figure 3.17. Now the label has gone away. Unfortunately, a number and a blank line are still left in place. See the example "Removing Numbers from the Table of Contents in HTML Output" in Chapter 16 for how to use the TEMPLATE procedure to get rid of the number and line in the HTML destination.

Figure 3.16

Table of Contents

1. Hours Billed by Customer
   - Summary statistics

Hours Billed by Customer

The MEANS Procedure

Analysis Variable : Hours Hours Worked		
Customer	Mean	Sum
Bricks and Mortar, Inc.	6.5	575.0
IPO.com	6.4	598.5
Smith & Smith	5.7	628.0
Sweat Shops Athletic	6.1	658.0
Virtual Co.	5.7	569.5

Figure 3.17

Table of Contents

1.
   - Summary statistics

Quarterly Report
Hours Billed by Customer

The MEANS Procedure

Analysis Variable : Hours Hours Worked		
Customer	Mean	Sum
Bricks and Mortar, Inc.	6.5	575.0
IPO.com	6.4	598.5
Smith & Smith	5.7	628.0
Sweat Shops Athletic	6.1	658.0
Virtual Co.	5.7	569.5

Changing Multiple Procedure Labels

The previous example showed a table of contents for a single procedure. But chances are that if you need to use ODS PROCLABEL to clean up your labels, it's because you have a bunch of procedures and a long table of contents.

The next example contains two PROC MEANS steps and one PROC PRINT step. An ODS PROCLABEL statement is used before the first PROC MEANS to change the label from "The MEANS Procedure" to "Hours Billed." A second ODS PROCLABEL statement is used to modify the label for the PROC PRINT output.

```
      ODS HTML BODY='body.html'
       CONTENTS='contents.html'
       FRAME='main.html';
title2 'Hours Billed by Customer';
ODS PROCLABEL 'Hours Billed';
proc means data=Billings nonobs mean sum maxdec=1;
   class CustomerName;
   var Hours;
run;
title2 'Hours Billed by Division';
proc means data=Billings nonobs mean sum maxdec=1;
    class Division;
    var Hours;
run;
title3 'Overtime Hours';
ODS PROCLABEL 'Overtime';
proc print data=Billings noobs;
   where Hours>8;
   var Hours Division WorkDate;
run;
ODS HTML CLOSE;
```

The output from this procedure is shown in Figure 3.18. Notice how the label in the table of contents for the first PROC MEANS step is changed to "Hours Billed." However, the second PROC MEANS step is still labeled "The MEANS Procedure." Unlike most ODS statements, ODS PROCLABEL applies only to the single procedure that immediately follows it in the code. Even though the second procedure is a PROC MEANS step just like the first one, ODS does not retain the PROCLABEL value.

To change the label for the second PROC MEANS step, add a second ODS PROCLABEL statement just before the PROC MEANS step.

Figure 3.18

Table of Contents

1. Hours Billed
   - Summary statistics
2. The Means Procedure
   - Summary statistics
3. Overtime
   - Data Set WORK.BILLINGS

Quarterly Report
Hours Billed by Customer

The MEANS Procedure

Analysis Variable : Hours Hours Worked		
Customer	**Mean**	**Sum**
Bricks and Mortar, Inc.	6.5	575.0
IPO.com	6.4	598.5
Smith & Smith	5.7	628.0
Sweat Shops Athletic	6.1	658.0
Virtual Co.	5.7	569.5

Changing the Title Bar

When you view HTML output in a Web browser, you see your SAS results in the browser main display area. You also see your SAS titles above your output in the browser window. There is a title bar at the top of the browser that also contains some text.

What you see in the title bar depends on your browser. By default, HTML created by ODS has the text “SAS Output” displayed in the title bar. Following that might be text such as “Microsoft Internet Explorer” or “Mozilla Firefox.”

For example, if you run the following code, you get the output shown in Figure 3.19.

```
      ODS HTML BODY='body.html';
title 'Quarterly Report';
      title2 'Hours Billed by Customer';
      proc means data=Billings nonobs mean sum maxdec=1;
        class CustomerName;
            var Hours;
      run;
      ODS HTML CLOSE;
```

Notice how the title created by the TITLE statement is shown in the main window. The title in the title bar of the browser is “SAS Output - Windows Internet Explorer.”

There’s nothing wrong with this. However, if you’d like to further customize your output, you can set it up so that the browser title bar matches the page title. All you need to do is add a new suboption, TITLE=, to your ODS HTML statement. Just follow the TITLE= with the text you want to display. The title text needs to be in quotes, and the whole TITLE= suboption needs to be enclosed in parentheses.

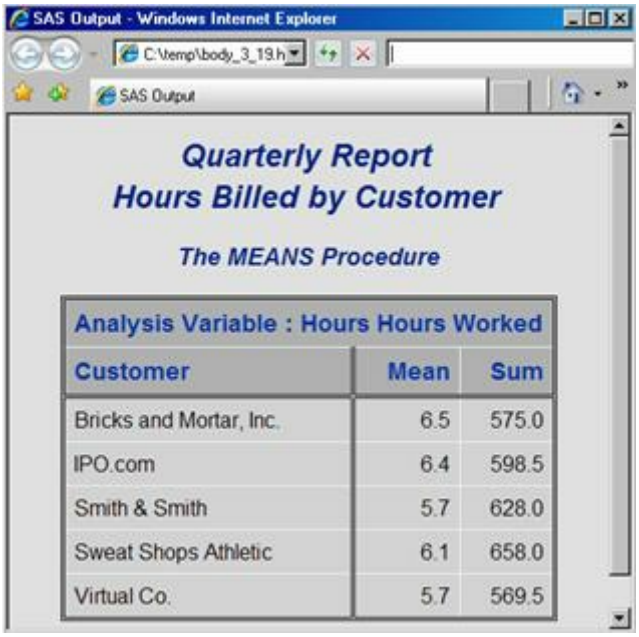
```
      ODS HTML BODY='body.html' (TITLE='Quarterly Report');
title "Quarterly Report";
      title2 'Hours Billed by Customer';
      proc means data=Billings nonobs mean sum maxdec=1;
        class CustomerName;
            var Hours;
      run;
      ODS HTML CLOSE;
```

The resulting Web page is shown in Figure 3.20. Notice that now the title bar reads “Quarterly
Report - Windows Internet Explorer.”

In this example, the title bar was set up to match the title in the SAS output. You could also change it to something else. Maybe on your Web site you like all of the title bars to display the name of your site. You could change the code to the following:

```
      ODS HTML BODY='body.html' (TITLE='MyWebSite.com');
```

Figure 3.19



SAS Output - Windows Internet Explorer

C:\temp\body_3_19.h

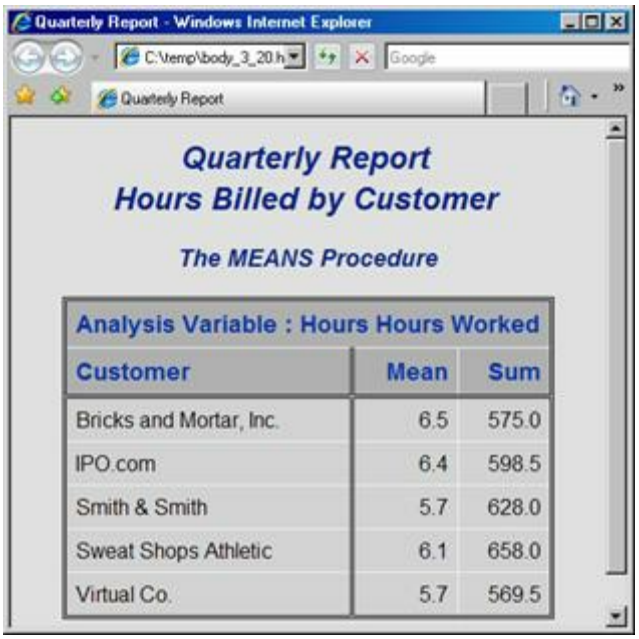
SAS Output

Quarterly Report
Hours Billed by Customer

The MEANS Procedure

Analysis Variable : Hours Hours Worked		
Customer	Mean	Sum
Bricks and Mortar, Inc.	6.5	575.0
IPO.com	6.4	598.5
Smith & Smith	5.7	628.0
Sweat Shops Athletic	6.1	658.0
Virtual Co.	5.7	569.5

Figure 3.20



Quarterly Report - Windows Internet Explorer

C:\temp\body_3_20.h

Google

Quarterly Report

Quarterly Report
Hours Billed by Customer

The MEANS Procedure

Analysis Variable : Hours Hours Worked		
Customer	Mean	Sum
Bricks and Mortar, Inc.	6.5	575.0
IPO.com	6.4	598.5
Smith & Smith	5.7	628.0
Sweat Shops Athletic	6.1	658.0
Virtual Co.	5.7	569.5

Creating Alternate HTML Formats

HTML comes in a variety of versions and sizes. For example, ODS HTML produces HTML
4.01-compliant tags in SAS®9. *Why would you want to switch HTML formats within SAS?* One reason is file size. The ODS HTML statement in SAS®9 produces a large inline style section. If you're creating a simple report, this probably does not matter. However, if your users have slow Internet connections, the file sizes can become prohibitively large. Three ways to create smaller files are to use ODS CHTML, ODS HTML with an external Cascading Style Sheet (CSS) file, or PHTML.

A CSS file contains instructions that define aspects of your HTML document such as color, font, and layout. When you want the same look for many documents, it might be more efficient to store the instructions in a CSS file and link to the CSS file for all the documents on a Web site. If you submitted the following PROC MEANS program multiple times, each time to a separate destination, your output files would be as described below:

```
<ODS invocation statement as shown below>
proc means data=Billings nonobs mean sum maxdec=1;
  class CustomerName;
      var Hours;
run;
ODS _ALL_ CLOSE;
```

`ODS HTML BODY='body.html';`	An ODS HTML statement produces an HTML 4.01 file that is displayed in Figure 3.21. This file is 30 KB in size.
`ODS CHTML BODY='chtml.html';`	An ODS CHTML statement produces a compact, minimal HTML file that is displayed in Figure 3.22. It creates the structure for the Web page, but leaves out all of the extra formatting tags. This file is only 1 KB in size.
`ODS HTML BODY='ht_css.html' stylesheet='default.css';`	Another ODS HTML statement, but one that uses the STYLESHEET= option, produces an HTML file that is displayed in Figure 3.23. It saves layout information in the separate CSS file DEFAULT.CSS. The HTML file is 3 KB in size and the CSS file is 28 KB.
`ODS PHTML BODY='phtml.html';`	Using an ODS PHTML statement produces a simple HTML file that is displayed in Figure 3.24. A "plain" HTML file can use up to twelve style elements. This file is 4 KB in size.

Since this example's HTML files are so small, there are negligible differences in page loading times. However, when working with more complex and lengthier output, it is important to consider how file size can make a difference in Web site performance. The tradeoff with using the simpler HTML formats like CHTML or PHTML is that you end up with a very basic Web page that does not take advantage of your ODS styles. Compare Figure 3.21 to Figures 3.22 and 3.23. Although the CSS file and the HTML file from Figure 3.23 take up 1 KB more space than the HTML file shown in Figure 3.21, the CSS file can be used over and over again.

Remember to verify with your webmaster whether you can deliver HTML pages that use sparse style on your Web site. Perhaps you will use one format, with CSS, on the company intranet and another format, with sparse style, on your external Web site. If this is the case, your webmaster might use XML or a scripting language to add style to the content as the pages are delivered.

Figure 3.21

The MEANS Procedure

Analysis Variable : Hours Hours Worked		
Customer	Mean	Sum
Bricks and Mortar, Inc.	6.5	575.0
IPO.com	6.4	598.5
Smith & Smith	5.7	628.0
Sweat Shops Athletic	6.1	658.0
Virtual Co.	5.7	569.5

Figure 3.22

The MEANS Procedure

Analysis Variable : Hours Hours Worked		
Customer	Mean	Sum
Bricks and Mortar, Inc.	6.5	575.0
IPO.com	6.4	598.5
Smith & Smith	5.7	628.0
Sweat Shops Athletic	6.1	658.0
Virtual Co.	5.7	569.5

Figure 3.23

The MEANS Procedure

Analysis Variable : Hours Hours Worked		
Customer	Mean	Sum
Bricks and Mortar, Inc.	6.5	575.0
IPO.com	6.4	598.5
Smith & Smith	5.7	628.0
Sweat Shops Athletic	6.1	658.0
Virtual Co.	5.7	569.5

Figure 3.24

The MEANS Procedure

Analysis Variable : Hours Hours Worked		
Customer	Mean	Sum
Bricks and Mortar, Inc.	6.5	575.0
IPO.com	6.4	598.5
Smith & Smith	5.7	628.0
Sweat Shops Athletic	6.1	658.0
Virtual Co.	5.7	569.5

Creating Multiple Copies of the Same Output

Using multiple ODS statements, you can create output for multiple destinations in one run. All you have to do is call ODS HTML, ODS RTF, and ODS PDF in separate statements before you run the procedure. You can also create multiple HTML versions of your output by repeating the ODS HTML statement. However, if you simply use two ODS HTML statements, one after the other (but with different filenames), the second invocation cancels out the first invocation.

This example shows how to build two Web pages with a single run through your procedures. All you need to do is number the ODS destination statements so that SAS can keep track of each version. Here is the syntax:

```
    ODS HTML(1) BODY='body1.html';
ODS HTML(2) BODY='body2.html';
*the procedures go here;
ODS HTML(1) CLOSE;
ODS HTML(2) CLOSE;
```

To see how this could come in handy, let's look at an example. Suppose you'd like to create two Web pages, one for your intranet and one for your external Web site. You want the intranet version to be informal, using the MINIMAL style, and to include an additional listing of detail data. You want the Web version to be more elaborate, using the Sasweb style, and not to include any detail data. The code is as follows (we've omitted the SAS TITLE statements so the example fits on one page):

```
ODS HTML(1) BODY='ForIntranet.html' Style=minimal;
ODS HTML(2) BODY='ForWeb.html' Style=sasweb;
proc tabulate data=billings missing;
  class division jobtype;
  var billableamt;
  tables jobtype all='Total',
        division*billableamt*(n*f=3. sum*f=dollar11.2);
run;
ODS HTML(2) CLOSE;
proc print data=billings n;
  where division='Reporting' and billableamt ge 1000;
  var jobtype employeename workdate customername hours billableamt
     quarter;
run;
ODS HTML(1) CLOSE;
```

In this example, two HTML statements are used to route output to two files, each specifying a different ODS style. Notice how both the HTML statements open their output files at the same time, but the second file is closed first, so that the PROC PRINT output goes only to the first HTML file. The two versions of the output are shown in Figure 3.25 and Figure 3.26. If you find the numbers confusing, there is another version of this syntax in which you can identify each HTML destination with a name instead of a number. The only syntax change is that you use the name, and if it's easier to understand, you can also preface the identifier with ID=.

ODS HTML(ID=*name*) BODY='filename.html';

Figure 3.25

Billable Amount by Division and Job Type

	Division											
	Analysis		Applications		Documentation		QA		Reporting		Systems	
	Amount Billable		Amount Billable		Amount Billable		Amount Billable		Amount Billable		Amount Billable	
	N	Sum	N	Sum	N	Sum	N	Sum	N	Sum	N	Sum
Job Type												
Consulting	41	$23,562.73	35	$19,179.82	36	$22,585.65	45	$27,408.19	37	$22,309.54	44	$25,842.47
Contract	15	$8,607.39	20	$12,191.57	17	$9,473.11	10	$4,128.29	12	$6,817.03	14	$9,147.52
In-house	26	$12,202.43	35	$19,497.90	27	$14,911.41	28	$18,533.30	25	$14,448.65	33	$18,456.36
Total	82	$44,372.55	90	$50,869.29	80	$46,970.17	83	$50,069.78	74	$43,575.22	91	$53,446.35

Billable Amount by Division and Job Type
For Internal Distribution Only
Reporting Division Billable Amount $1000+

Obs	JobType	EmployeeName	WorkDate	CustomerName	Hours	BillableAmt	Quarter
8	In-house	Platt	09/05/2010	Bricks and Mortar, Inc.	11.0	$1025.85	2010-3
10	Consulting	Smith	03/21/2009	Smith & Smith	11.0	$1022.80	2009-1
284	Consulting	Jones	09/21/2009	IPO.com	11.0	$1024.82	2009-3
353	In-house	Platt	04/08/2009	Smith & Smith	11.0	$1025.80	2009-2
423	Contract	Haworth	07/13/2009	IPO.com	11.0	$1021.81	2009-3
N = 5							

Figure 3.26

Billable Amount by Division and Job Type

	Division											
	Analysis		Applications		Documentation		QA		Reporting		Systems	
	Amount Billable		Amount Billable		Amount Billable		Amount Billable		Amount Billable		Amount Billable	
	N	Sum	N	Sum	N	Sum	N	Sum	N	Sum	N	Sum
Job Type												
Consulting	41	$23,562.73	35	$19,179.82	36	$22,585.65	45	$27,408.19	37	$22,309.54	44	$25,842.47
Contract	15	$8,607.39	20	$12,191.57	17	$9,473.11	10	$4,128.29	12	$6,817.03	14	$9,147.52
In-house	26	$12,202.43	35	$19,497.90	27	$14,911.41	28	$18,533.30	25	$14,448.65	33	$18,456.36
Total	82	$44,372.55	90	$50,869.29	80	$46,970.17	83	$50,069.78	74	$43,575.22	91	$53,446.35

Exploring Links and Anchors

You don't have to understand HTML code to generate output with a table of contents or a table of pages. However, if you understand the basics of how the code works, it makes it easier to customize your output. The table of contents file has to communicate via the browser with the main body file. When you click on an item in the table of contents, the browser uses links and references to determine where to go in the body file. "Exploring the Frame File" earlier in this chapter shows how the table of contents and body pages are set up within a frame page. This example shows how the links between the two pages work.

The following code produces a simple Web page with a table of contents that we can use to explore how this works. The resulting output is shown in Figure 3.27. In this screen view, assume you have just clicked the "Extreme Observations" label, so that portion of the output is displayed in the right-hand frame.

```
      ODS HTML BODY='body.html'
       CONTENTS='contents.html'
       FRAME='main.html';
proc univariate data=Billings plots;
   var Hours;
run;
ODS HTML CLOSE;
```

How did the browser know what to do when you clicked the "Extreme Observations" label? The answer is hyperlinks and anchors. When SAS builds the table of contents file, it uses HTML anchor tags <A> and </A> and embeds links to the body page using the HREF= attribute. If you look at Figure 3.28, you will see a series of anchor tags in the HTML code for the table of contents. Each anchor tag contains an HREF= attribute that points to a location in the body file. Looking more closely at the anchor tag for the "Extreme Observations" label in the table of contents, you can see that it points to the HREF "body_links.html#IDX4". Also notice the TARGET= attribute. This tells the browser where to display the linked information (in this case, in the frame called "body"). What's the "#IDX4" about? This is how the browser knows to jump to the part of the body file that contains the extreme observations. SAS has seeded the body file with a numbered series of anchors, one before each table in the file. The table of contents file uses these numbered anchor names to enable the links to go to the right part of the file.

Figure 3.29 displays a portion of the body file to show how this works. In this case, the HTML for the output itself has been deleted so you can see the anchor tags. SAS has inserted a pair of anchor tags before each output object in the results. Looking at the anchor tag that comes right before the extreme observations results, you can see that it is named "IDX4". So if the table of contents links to "body_links.html#IDX4", the browser knows to display the body file, and to scroll down to the anchor named "IDX4". (By the way, the " " between the anchor tags just creates a single non-breaking space. Anchor tags have to be wrapped around something, but you don't want an extra bit of text displayed in the browser, so SAS uses a blank space.)

To learn more about HTML, searching the Internet for "HTML tutorials" should turn up a number of sites that offer online or downloadable tutorials.

Figure 3.27

Table of Contents

1. The Univariate Procedure
   - Hours
     - Moments
     - Basic Measures of Location and Variability
     - Tests For Location
     - Quantiles
     - Extreme Observations
     - Plots

Extreme Observations			
Lowest		Highest	
Value	Obs	Value	Obs
1	475	11	284
1	469	11	353
1	466	11	400
1	426	11	423
1	334	11	445

Figure 3.28

```
<BODY>
Table of Contents<HR size=3>
The Univariate Procedure<br>Hours<br>
<A HREF="body_links.html#IDX" TARGET="body">Moments</a><br>
<A HREF="body_links.html#IDX1" TARGET="body">Basic Measures of Location and
Variability</a>
<A HREF="body_links.html#IDX2" TARGET="body">Tests For Location</a>
<A HREF="body_links.html#IDX3" TARGET="body">Quantiles</a>
<A HREF="body_links.html#IDX4" TARGET="body">Extreme Observations</a>
<A HREF="body_links.html#IDX5" TARGET="body">Plots</a><br>
</BODY>
```

Figure 3.29

```
(Heading tags and body tag go here.)
<A NAME="IDX"> </A>
(code for displaying moments goes here)
<A NAME="IDX1"> </A>
(code for displaying basic measures goes here)
<A NAME="IDX2"> </A>
(code for displaying tests for location goes here)
<A NAME="IDX3"> </A>
(code for displaying quantiles goes here)
<A NAME="IDX4"> </A>
(code for displaying extreme observations goes here)
<A NAME="IDX5"> </A>
(Code for displaying plots and ending body and html tags go here.)
```

Using Links and Anchors to Link Your Results to Your Web Site

By default, SAS creates anchor tag names starting with IDX and then adds a number after each subsequent tag. This is nice because the structure and cross links are set up with no effort required by the programmer. In addition, this behavior is predictable, so you can take advantage of it to integrate your output into a Web site.

For example, let's say you want to add the PROC UNIVARIATE output from the previous example to a Web site that contains other information about billing policies and hours. Figure 3.30 shows this Web page. Our company, the fictitious Computer Consulting Corporation, has this page already set up. It has links to other files for policies and forms.

One way to add our PROC UNIVARIATE output from the previous example would be to link to the frame file. Then we'd have to show the whole page, with a table of contents and a body. It would be nicer to just add a couple of links to our existing page, pointing to just the relevant parts of the report. For example, we could have a Web page like the one in Figure 3.31, which has a new section titled "Reports," with links to two parts of the SAS results.

This is very easy to do. First, you need to run the SAS code to generate a body file. Even if you don't specify a table of contents, SAS puts anchor tags using the "IDX" naming convention in your output.

```
      ODS HTML BODY='body.html';
proc univariate data=Billings plots;
   var Hours;
run;
ODS HTML CLOSE;
```

Then, after creating the HTML file, all you have to do is upload it to your Web site and add two new links to the HTML code for your existing Web page. To add a hyperlink to the descriptive statistics at the top of the univariate output, you add the following HTML code to your file:

```
      <a href="http://www.compconscorp.com/billing/body.html#idx">
Descriptive Statistics</a>
```

This code assumes that you've uploaded the "body.html" file you created to your Web site, and placed it in the "billing" directory. This code is displayed in the Web browser as the text "Descriptive Statistics," and when you click on the text, the browser displays your SAS output showing the Moments section at the top. The code for adding a hyperlink to the extreme observations is similar:

```
      <a href="http://www.compconscorp.com/billing/body.html#idx4">
Extremes</a>
```

In this case, we knew that the anchors were called "IDX" and "IDX4" because we'd been looking at the HTML code, but you can figure out the anchor names without looking at the

HTML. Just count the output objects. For example, to link to the third object, specify the "IDX2" anchor. In this case, we wanted the first and fifth objects, so the anchors were "IDX" and "IDX4".

Figure 3.30

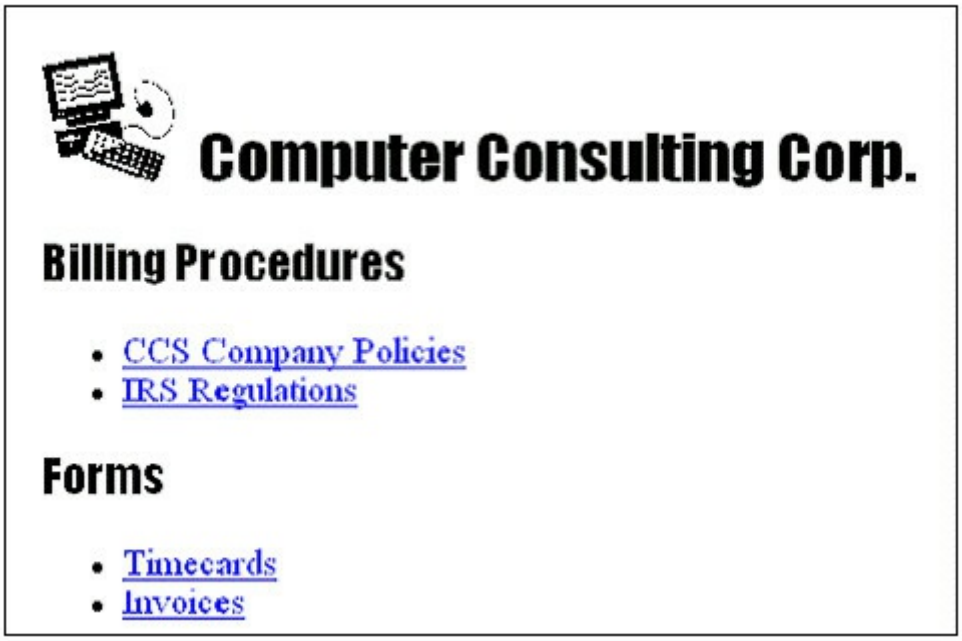
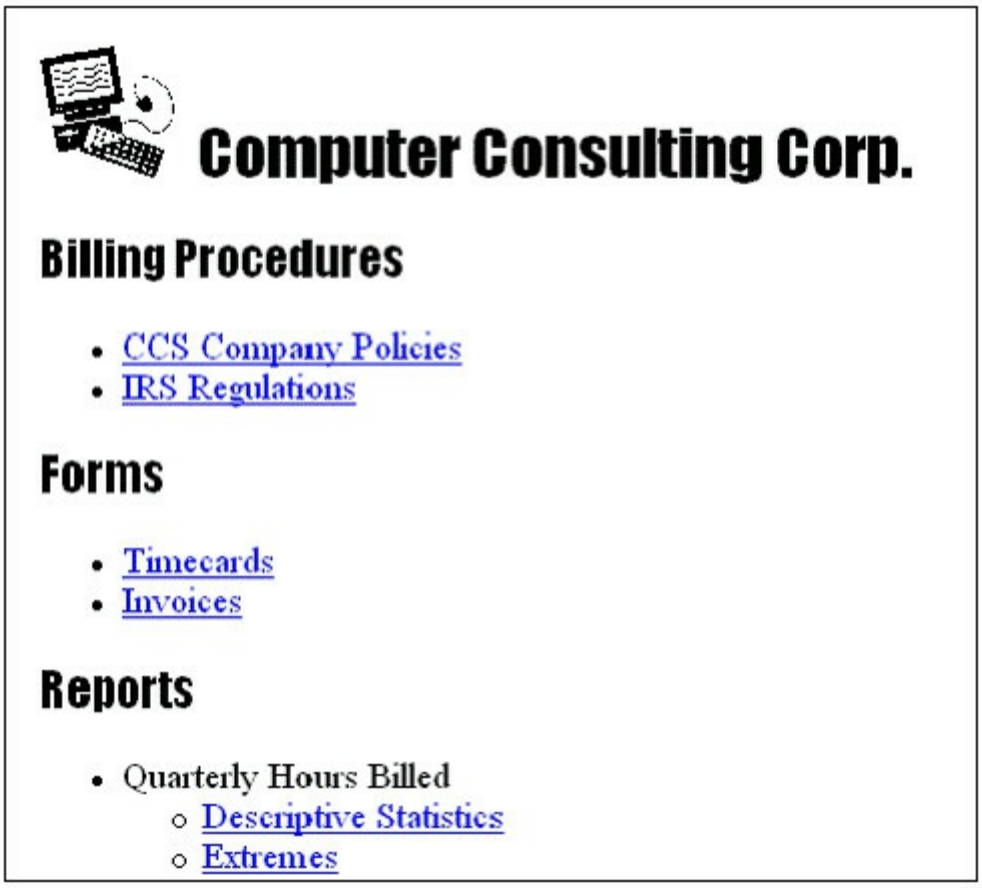


Figure 3.31



Adding HTML Code to Customize Your Output

The previous examples in this chapter have introduced the basics of HTML tags. You might find yourself asking why you should bother learning HTML tags when SAS can create them for you. The reason is that you can use your knowledge of HTML tags to add tags to your code to enhance your output. And you don't have to edit the HTML file to do so. You can use SAS to insert the extra tags when the HTML output is created.

As an example, we'll modify the simple output shown in Figure 3.32. Notice how both the title and footnote are shown in the same font size. Also, notice how all of the column headings are the same size and color. It would be nice for the title to be large and the footnote to be small and unobtrusive. Also, it would be nice to do something to the column heading for overtime hours to call attention to this important data. The following code inserts HTML tags into the TITLE, FOOTNOTE, and LABEL statements to change the font appearance in each location. SAS takes these tags and passes them along to the output file, where they can be interpreted by a browser.

Unfortunately, SAS also passes the tags along to other non-HTML output destinations, where they appear as text in your title and footnote. Modifying your output with tags works only with the HTML destination. Therefore, you will want to close any other open destinations when using this technique. See Chapter 15, "Modifying Output Fonts," for information about how to use the TEMPLATE procedure to modify fonts and colors for all destinations by changing the style definition.

The following program includes three pairs of HTML tags. First, the <H1> and </H1> tags assign "Heading Style 1" ("H1") to the title. This style requests the largest heading available from the browser. The second pair of tags, <FONT SIZE=2> and </FONT>, specify a smaller font for the footnote text. Finally, the third pair of tags, <I> and </I>, italicize the column heading for HOURS.

```
      ODS HTML BODY='overtime.html';
title '<H1>Overtime Billing for Contract Job Type</H1>';
footnote '<FONT SIZE=2>Overtime defined as >10 hours</FONT>';
proc print data=Billings noobs label;
   where Hours>8;
   label Hours='<I>Overtime Hours</I>';
   var WorkDate JobType Hours;
run;
ODS HTML CLOSE;
```

The results are shown in Figure 3.33. This example is a simple illustration of customizing your output by adding HTML tags. With dozens of other HTML tags and CSS style properties available, you have the capability to truly customize the appearance of your output.

Another way to add HTML formatting to your titles and footnotes is to use the LINK=, FONT=, COLOR=, HEIGHT=, BOLD, ITALIC, and JUST= options available with the TITLE and FOOTNOTE statements. ODS builds the HTML code for you when you use these options. Your tradeoff is decreased control. You must decide whether to allow ODS to build the HTML tags or to use custom HTML tags to give you total control over tag appearance. Your choice depends on your level of HTML expertise and your Web site standards.

Figure 3.32

Overtime Billing for Contract Job Type

Employee Name	Date Work Completed	Job Type	Overtime Hours
Jones	01/09/2009	Contract	10.5
Smith	05/09/2009	Contract	10.5
Haworth	07/06/2009	Contract	11.0
Haworth	07/13/2009	Contract	11.0
Smith	08/24/2009	Contract	10.5
Platt	07/27/2010	Contract	10.5
Smith	09/30/2010	Contract	10.5

Overtime defined as >10 hours

Figure 3.33

Overtime Billing for Contract Job Type

Employee Name	Date Work Completed	Job Type	*Overtime Hours*
Jones	01/09/2009	Contract	10.5
Smith	05/09/2009	Contract	10.5
Haworth	07/06/2009	Contract	11.0
Haworth	07/13/2009	Contract	11.0
Smith	08/24/2009	Contract	10.5
Platt	07/27/2010	Contract	10.5
Smith	09/30/2010	Contract	10.5

Overtime defined as >10 hours

Limiting Inclusion of HTML Tags in Output

When you add HTML code to your output, as in the previous example, those same tags turn up, usually as text, in every output destination. For example, if the code in the previous example were used to generate RTF output, the HTML tags would be passed through as text to the RTF destination, creating the table shown in Figure 3.34. The word processor does not recognize the HTML tags as anything but text.

There are two ways you could solve the problem. The first method is to run the program twice. The first submission creates your HTML table, and the second submission creates your RTF table after you remove the embedded HTML tags from the code.

For the second method, you only need to run the program once because you modify the code to identify the HTML tags in a way so that ODS does not pass them to the RTF destination. You can use an inline formatting command to set up the tags for the HTML destination only.

You start by using the ODS ESCAPECHAR statement. The escape character you specify with this statement tells ODS that a formatting command follows. The next example in this chapter "Applying Inline Formatting" describes this process in more detail.

```
ODS ESCAPECHAR="^";
```

When specifying formatting instructions, you start with the escape character and follow it with your formatting instruction and text to be formatted enclosed in brackets. Preceding the instruction and text, you can add the DEST option and enclose destinations in brackets. The destinations enclosed in brackets are the only ones for which the following formatting instruction should apply. To use it with HTML tags, the syntax is as follows:

```
^{DEST [HTML]the html tags go here}"
```

The program in the previous example is modified below so that the HTML tags are passed only to the HTML destination. Here, the revised code for the title is shown:

```
title '^{DEST [HTML] <H1>}Overtime Billing for Contract Job Type^{DEST [HTML]</H1>}';
```

There are two ^{DEST [HTML] } specifications. The first one blocks out the beginning of the HTML tag, and the second one blocks out the end of the HTML tag. The text between the two anchor tags is left alone and is passed to both the HTML and RTF destinations.

The result of this new code is that the HTML output, shown in Figure 3.34, is unchanged. But the RTF output now excludes the HTML tags, as shown in Figure 3.35.

Figure 3.34

<H1>Overtime Billing for Contract Job Type</H1>

Employee Name	Date Work Completed	Job Type	<I>Overtime Hours</I>
Jones	01/09/2009	Contract	10.5
Smith	05/09/2009	Contract	10.5
Haworth	07/06/2009	Contract	11.0
Haworth	07/13/2009	Contract	11.0
Smith	08/24/2009	Contract	10.5
Platt	07/27/2010	Contract	10.5
Smith	09/30/2010	Contract	10.5

<FONT SIZE=2>Overtime defined as >10 hours</FONT>

Figure 3.35

Overtime Billing for Contract Job Type

Employee Name	Date Work Completed	Job Type	Overtime Hours
Jones	01/09/2009	Contract	10.5
Smith	05/09/2009	Contract	10.5
Haworth	07/06/2009	Contract	11.0
Haworth	07/13/2009	Contract	11.0
Smith	08/24/2009	Contract	10.5
Platt	07/27/2010	Contract	10.5
Smith	09/30/2010	Contract	10.5

Overtime defined as >10 hours

Applying Inline Formatting

As shown in the previous example, inline formatting enables you to make simple formatting changes in specific parts of your output, and for specific output destinations. Inline formatting allows you to specify the style and layout characteristics of almost any cell or line in your output.

For example, if you'd like to add footnote references to some of the labels in your output, you can use inline formatting to add the footnote numbers as superscripts. The following program demonstrates this technique by adding footnotes to the row labels of PROC MEANS output.

The first thing you need to do to use inline formatting is to designate an escape character to be used to identify inline formatting. It needs to be a character that is not used anywhere else in your output or program. Suggested characters include ~, ^, #, and @.

```
ODS ESCAPECHAR="^";
```

Now you can use this character to add superscripts to your format labels. The syntax for applying inline formatting always starts with the escape character. Then, for a superscript, you enclose the SUPER function name and the text to superscript in curly brackets ({ }), as shown below.

```
    proc format;
  value superfmt 1='Control'
                 2='Low Dose^{super 1}'
                 3='High Dose^{super 2}';
run;
```

If you apply the SUPERFMT format to CLASS variable TX in a PROC MEANS step that analyzes the data set CLINICAL, you get the output shown in Figure 3.36. Here is the PROC MEANS step:

```
options nocenter;
proc means data=clinical mean std maxdec=1;
  class tx;
  var postsbp postdbp;
  format tx superfmt.;
run;
```

With footnote references in the table, the next step is to add the footnotes themselves. Inline formatting in the following FOOTNOTE statements generates superscripts for the two footnote numbers.

```
    footnote "^{super 1}2 mL/kg.";
footnote2 "^{super 2}5 mL/kg";
```

Figure 3.37 shows the new results with footnotes added to the report and with OPTIONS NOCENTER in effect. Although the footnote superscripts match the table, the font size is so large that they look out of proportion to the table. To fix this, you can use the inline formatting STYLE function. Start with the escape character and enclose in curly brackets ({}) the STYLE function name and the style attributes that you want to modify. The expression for

each attribute and its new value is enclosed in square brackets ([]). You can nest functions as well, which the following FOOTNOTE statements demonstrate. The FONT_SIZE=2 attribute is applied to all the text in each footnote, while superscripting is applied only to the footnote numbers. The output with these new footnotes is shown in Figure 3.38.

```
footnote "^{style [font_size=2] ^{super 1}2 mL/kg.}";
footnote2 "^{style [font_size=2] ^{super 2}5 mL/kg}";
```

Figure 3.36

Treatment Group	N Obs	Variable	Label	Mean	Std Dev
Control	76	postsbp	Post-Treatment Systolic BP	111.4	8.1
		postdbp	Post-Treatment Diastolic BP	79.0	10.9
Low Dose[1]	65	postsbp	Post-Treatment Systolic BP	113.0	7.0
		postdbp	Post-Treatment Diastolic BP	80.0	10.6
High Dose[2]	79	postsbp	Post-Treatment Systolic BP	109.7	7.7
		postdbp	Post-Treatment Diastolic BP	64.0	15.7

Figure 3.37

Treatment Group	N Obs	Variable	Label	Mean	Std Dev
Control	76	postsbp	Post-Treatment Systolic BP	111.4	8.1
		postdbp	Post-Treatment Diastolic BP	79.0	10.9
Low Dose[1]	65	postsbp	Post-Treatment Systolic BP	113.0	7.0
		postdbp	Post-Treatment Diastolic BP	80.0	10.6
High Dose[2]	79	postsbp	Post-Treatment Systolic BP	109.7	7.7
		postdbp	Post-Treatment Diastolic BP	64.0	15.7

[1]2 mL/kg.
[2]5 mL/kg

Figure 3.38

Treatment Group	N Obs	Variable	Label	Mean	Std Dev
Control	76	postsbp	Post-Treatment Systolic BP	111.4	8.1
		postdbp	Post-Treatment Diastolic BP	79.0	10.9
Low Dose[1]	65	postsbp	Post-Treatment Systolic BP	113.0	7.0
		postdbp	Post-Treatment Diastolic BP	80.0	10.6
High Dose[2]	79	postsbp	Post-Treatment Systolic BP	109.7	7.7
		postdbp	Post-Treatment Diastolic BP	64.0	15.7

[1]2 mL/kg.

[2]5 mL/kg

Adding JavaScript to Your Output

Most Web sites with special effects like buttons that change color when selected or customized greetings to visitors are not created using just HTML. They use languages such as Java, JavaScript, and Perl to achieve those effects. If you know how to write scripts, or you already have scripts you use on your Web site, you can add these scripts to your ODS output.

This example shows how to add JavaScript to your ODS output by creating a Web page with a hit counter. It assumes that you've got two scripts written in JavaScript. One sets up the functions used to calculate the value of the hit counter, and the other contains the code used to display the hit counter.

The first step is to define two macro variables, HEADTEXT and FOOTER, to each hold one of the two scripts. Notice that any quotation marks in the script are doubled. The SAS %QUOTE macro function surrounds the script tag to preserve the quotation marks.

```
%let headtext=%quote(<SCRIPT language=""Javascript"" src=""counter.js"" type=""text/javascript""> </SCRIPT>);

%let footer=%quote(<SCRIPT language=""Javascript"" src=""counter2.js"" type=""text/javascript""></SCRIPT>);
```

The first script, called "COUNTER.JS", contains the functions that compute the value of the hit counter. This script needs to end up in the header of the HTML output. To do this, add the script code to the Web page between the <HEAD> and </HEAD> tags. Fortunately, ODS makes this action easy because all you have to do is add the HEADTEXT= option to your ODS HTML statement, as in the following code. The value supplied to HEADTEXT= is a reference to macro variable HEADTEXT, which specifies the SCRIPT tag that calls COUNTER.JS.

```
ODS HTML PATH='.' (URL=NONE) BODY='counter.html'
 HEADTEXT="&headtext";
footnote justify=right "&footer";
proc freq data=clinical;
  tables tx;
     format tx txft.;
run;
ODS HTML CLOSE;
```

The second script, "COUNTER2.JS", contains the script code that displays the hit counter. The goal is to display the hit counter at the bottom of the Web page. To do this, place a reference to macro variable FOOTER in a FOOTNOTE statement. Macro variable FOOTER contains the script tag that calls COUNTER2.JS. The JUSTIFY=RIGHT option right-justifies the footnote. The first time the Web page is viewed, it shows a hit count of 1, as in Figure 3.39. The next time the page is visited, the hit counter increments by 1, as shown in Figure 3.39.

You can use this technique to add any scripts you need to enhance your ODS output. Just load the scripts themselves into the header using HEADTEXT=, and add the script that calls

the other scripts into the body using the TITLE or FOOTNOTE statement. For a short script, you could skip the step of assigning the script to a macro variable and instead specify the script directly as the value for the style attribute.

Figure 3.39

Treatment Group				
tx	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Control	76	34.55	76	34.55
Low Dose	65	29.55	141	64.09
High Dose	79	35.91	220	100.00

1 Hit

Figure 3.40

Treatment Group				
tx	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Control	76	34.55	76	34.55
Low Dose	65	29.55	141	64.09
High Dose	79	35.91	220	100.00

2 Hits

You can also load the script by changing the PREHTML or POSTHTML style attributes in a custom style template. This technique might be useful for the type of Javascript that would be used on every HTML file that you create with a specific ODS style.

Creating and Using Cascading Style Sheets

Many Web developers choose to control the formatting of their Web pages using Cascading Style Sheets (CSS). These are separate files that are used by browsers when they render a page to control things like colors, fonts, alignment, and borders. Separating the formatting information from the Web page reduces the file size of the Web page, which in turn reduces the time it takes the page to load.

As already discussed, your webmaster will not only have an opinion about the flavor of HTML you choose, but will also know whether you are required to use CSS technology when placing reports on a company Internet or intranet site. Your webmaster can also give you the correct URL if your CSS file is stored on a company Web server. But of most interest to you is how to instruct ODS about the type of CSS you want to use. Consider the following code:

```
ODS HTML(ID=1) BODY='make_externalcss.html'
   STYLESHEET='default.css';
ODS HTML(ID=2) BODY='external_sasweb.html'
   STYLESHEET='sasweb.css' STYLE=SASWEB;
ODS HTML(ID=3) BODY='use_externalcss.html'
   STYLESHEET=(URL='corpstyle.css');
proc means data=clinical mean std maxdec=2;
  class gender;
  var chgsbp chgdbp;
run;
ODS _ALL_ CLOSE;
```

The first two invocations instruct ODS to both create and use a CSS file in the creation of output. However, the third invocation instructs ODS to use an existing CSS file.

ODS HTML(ID=1) or ODS HTML(ID=2): To produce an external CSS file and to have ODS insert the appropriate <LINK> tag into the HTML output file, you use the syntax shown in either the ODS HTML (ID=1) statement or the ODS HTML (ID=2) statement. The output from ID=1 statement and some of the underlying HTML code is shown in Figure 3.41, while the output from the ID=2 statement with its underlying HTML code is shown in Figure 3.42. For your convenience, you can compare just the <LINK> tags that were inserted into the HTML files in Figure 3.44.

ODS HTML(ID=3): The ODS HTML(ID=3) statement assumes that you already have an external CSS file and that the corporate CSS file has a definition to make the headers have a black background and a white foreground.

```
. .Header{
  font-family: Arial, Helvetica, sans-serif;
  font-size: x-small; font-weight: bold; font-style: normal;
  color: #FFFFFF; background-color: #000000;}
```

Figure 3.43 shows the <LINK> tag that was inserted for the CORPSTYLE.CSS reference in the HTML file. The figure also shows the HTML output and some of the underlying HTML code.

New with SAS 9.2 is the ability to use CSS files with both the RTF and PDF destinations. For an example of that capability, refer to Chapter 18, “Doing More with Styles and ODS.”

Figure 3.41

Gender	N Obs	Variable	Label	Mean	Std Dev
Female	108	chgsbp	Change in Systolic BP	-25.29	22.28
		chgdbp	Change in Diastolic BP	-20.12	21.82
Male	112	chgsbp	Change in Systolic BP	-26.67	22.79
		chgdbp	Change in Diastolic BP	-16.79	20.16

make_externalcss.html - Notepad

File Edit Format View Help

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta name="Generator" content="SAS Software Version 9.2, see www.sas.com">
<meta http-equiv="Content-type" content="text/html; charset=windows-1252">
<title>SAS Output</title>
<link rel="stylesheet" type="text/css" href="default.css">
<style type="text/css">
```

Figure 3.42

Gender	N Obs	Variable	Label	Mean	Std Dev
Female	108	chgsbp	Change in Systolic BP	-25.29	22.28
		chgdbp	Change in Diastolic BP	-20.12	21.82
Male	112	chgsbp	Change in Systolic BP	-26.67	22.79
		chgdbp	Change in Diastolic BP	-16.79	20.16

external_sasweb.html - Notepad

File Edit Format View Help

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta name="Generator" content="SAS Software Version 9.2, see www.sas.com">
<meta http-equiv="Content-type" content="text/html; charset=windows-1252">
<title>SAS Output</title>
<link rel="stylesheet" type="text/css" href="sasweb.css">
```

Figure 3.43: HTML Output Using the CORPSTYLE CSS

Gender	N Obs	Variable	Label	Mean	Std Dev
Female	108	chgsbp	Change in Systolic BP	-25.29	22.28
		chgdbp	Change in Diastolic BP	-20.12	21.82
Male	112	chgsbp	Change in Systolic BP	-26.67	22.79
		chgdbp	Change in Diastolic BP	-16.79	20.16

use_externalcss.html - Notepad

File Edit Format View Help

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta name="Generator" content="SAS Software Version 9.2, see www.sas.com">
<meta http-equiv="Content-type" content="text/html; charset=windows-1252">
<title>SAS Output</title>
```

Figure 3.44: <LINK> Tag Generated for All Examples

(ID=1)	`<link rel="stylesheet" type="text/css" href="default.css">`
(ID=2)	`<link rel="stylesheet" type="text/css" href="sasweb.css">`
(ID=3)	`<link rel="stylesheet" type="text/css" href="corpstyle.css">`

Creating Drill-Downs within Output Tables

When you create a Web page with a table of contents, SAS builds a structure to help users navigate through your output. But sometimes you want your output to be more directly linked together. If users are looking at a summary table with a number of variables, it would be nice if they could jump directly to univariate statistics for each variable.

What you want users to be able to do is *drill down* to detailed statistics from data summarized in a table. This type of table is easy to create if you know a little about HTML. What you do is create a table and then build hyperlinks into your variable labels.

The following code shows how this works with a table created by PROC TABULATE. The label for each of the analysis variables has an anchor tag. The anchor tag contains the URL for the Web page with the detailed statistics, and the tag includes the text to display in the PROC TABULATE output. For example, the following code generates the label "Baseline DBP" for the variable BASEDBP. This label is a hyperlink to the Web page univ_basedbp.html.

```
      ODS HTML FILE='main.html';
proc tabulate data=clinical f=8.1;
  var basesbp basedbp chgsbp chgdbp;
  class tx;
  table basedbp='<a href="univ_basedbp.html">Baseline DBP</a>'
        basesbp='<a href="univ_basesbp.html">Baseline SBP</a>'
        chgdbp='<a href="univ_chgdbp.html">Change in DBP</a>'
        chgsbp='<a href="univ_chgsbp.html">Change in SBP</a>',
        mean*tx=' ';
run;
ODS HTML CLOSE;
```

Designing the main table and determining the labels that will be hyperlinks is the first step. Before running the PROC TABULATE step, however, it is necessary to create the Web pages referenced by the anchor tags in each variable label. An easy way to generate the four outputs is with a macro program. The macro program UNIV that follows executes a PROC UNIVARIATE step for the variable specified by the value of macro variable parameter VARNAME. The variable name is also inserted in the name of the HTML output file.

```
      %macro univ(varname);
  ODS HTML FILE="univ_&varname..html";
  proc univariate data=clinical;
    var &varname;
  run;
  ODS HTML CLOSE;
%mend univ;
%univ(basedbp)
%univ(basesbp)
%univ(chgdbp)
%univ(chgsbp)
```

After this code has been run, the hyperlinks in the main table will function. Figure 3.45 shows the main table with the hyperlinks. When you click on the link "Baseline DBP," you are taken to the Web page univ_basedbp.html, the first table of which is shown in Figure 3.46.

Figure 3.45

	Mean		
	Control	Low Dose	High Dose
Baseline DBP	94.4	94.6	88.5
Baseline SBP	138.7	136.8	136.3
Change in DBP	-15.4	-14.6	-24.5
Change in SBP	-27.3	-23.8	-26.6

Variable: basedbp (Baseline Diastolic BP)

Moments			
N	220	Sum Weights	220
Mean	92.3409091	Sum Observations	20315
Std Deviation	16.424927	Variance	269.778227
Skewness	0.23351007	Kurtosis	-1.0546829
Uncorrected SS	1934987	Corrected SS	59081.4318
Coeff Variation	17.7872702	Std Error Mean	1.10736835

Figure 3.46

This example used a macro program to build the Web pages for each of the drill-downs in the main table. With a little ingenuity, you can write a macro program to build both the main table and the drill-downs.

Creating Output with Just the Results

When ODS creates a Web page, it includes not just your results, but also all of the HTML tags that are needed to tell a browser how to display the page. This includes the tags for things like the text for the title bar, a number of META tags that describe the output, and some script outlines that are not used by most types of ODS output. It also includes the HTML tags that begin and end the page.

All of this is useful if your ODS Web page will be used on its own. But if you want to embed your ODS results in an existing Web page, all of these extra tags get in the way. All you really need is the HTML for the part of the page that contains your results.

To request that ODS produce just the part of the page with your results, without the other HTML that comes before and after, you can use two suboptions that apply to your body file:

```
      NO_TOP_MATTER       (alternate abbreviation: NOTOP)
NO_BOTTOM_MATTER     (alternate abbreviation: NOBOT)
```

To use these suboptions in your code, put them in parentheses after the BODY= option in your ODS HTML statement, as in the following example:

```
      ODS HTML BODY='TreatGroup.html' (NOTOP NOBOT);
proc freq data=clinical;
   tables tx / nocum;
         format tx txft.;
run;
ODS HTML CLOSE;
```

The result is the HTML file shown in Figure 3.47.1_ It contains table tags, but no <HTML>, <HEAD>, <TITLE>, or <BODY> tags. It has all of the information needed to display the table, but no more. Without top or bottom tags, the output might display as shown in Figure 3.48. However, if you just open the HTML file created by this procedure directly in a browser, it might not display at all. Some browsers can figure out to add the extra tags needed to display these results; others return an error.

If your webmaster incorporates this code within an existing Web page, or you copy and paste this code into an existing Web page (using the default SAS style), then the table appears as shown in Figure 3.49.

Figure 3.47

```
<table class="Table" cellspacing="1" cellpadding="7" rules="groups" frame="box"
summary="Procedure Freq: One-Way Frequencies">
<thead>
<tr>
<th class="c b Header" colspan="3" scope="colgroup">Treatment Group</th>
</tr>
<tr>
<th class="r b Header" scope="col">tx</th>
<th class="r b Header" scope="col">Frequency</th>
<th class="r b Header" scope="col"> Percent</th>
</tr>
</thead>
<tbody>
<tr>
<th class="r RowHeader" scope="row">Control</th>
<td class="r Data">78</td>
<td class="r Data">35.45</td>
</tr>
<tr>
<th class="r RowHeader" scope="row">Low Dose</th>
<td class="r Data">76</td>
<td class="r Data">34.55</td>
</tr>
<tr>
<th class="r RowHeader" scope="row">High Dose</th>
<td class="r Data">66</td>
<td class="r Data">30.00</td>
</tr>
</tbody>
</table>
```

Figure 3.48 Possible Browser View

The FREQ Procedure

Treatment Group		
tx	**Frequency**	**Percent**
Control	76	34.55
Low Dose	65	29.55
High Dose	79	35.91

Figure 3.49
Table Incorporated into Existing Web Page

The FREQ Procedure

Treatment Group		
tx	Frequency	Percent
Control	76	34.55
Low Dose	65	29.55
High Dose	79	35.91

Appending to an Existing HTML File

If you want to include a number of procedure results in the same Web page, you can list a number of procedures between your ODS HTML statements. But what if you want to run the procedures at different times, or on different servers?

The NOTOP and NOBOT suboptions can also be used to start a Web page in one run, and then finish it in another run. The way it works is that you specify the NOBOT suboption in the ODS HTML statement for the first run. For the second run, you specify the NOTOP suboption in another ODS HTML statement and include a FILENAME statement with the MOD option, which allows appending one file to another.

The NOBOT suboption in the ODS HTML statement in the first run creates the HTML top matter in the output file and does not create the bottom matter. The NOTOP suboption on the second HTML statement does not create the HTML top matter from the second program in the HTML output file, but does create the bottom matter.

The code for the first run is shown below. In this example, this first procedure is run on a UNIX server. Figure 3.50 presents the results from this first run.

```
    ODS HTML BODY='/tmp/trial1/results/Gender.html' (NOBOT);
proc means data=clinical mean std maxdec=2;
  class gender;
  var chgsbp chgdbp;
run;
ODS HTML CLOSE;
```

This code creates an HTML file with all of the required heading tags and your first set of procedure results. But it leaves off the extra tags at the bottom.

The following PROC FREQ step is submitted under Windows and its results are appended to the file started by the preceding UNIX submission. The example starts with a FILENAME statement that associates a fileref to the HTML file. The MOD option specifies that the output from the PROC FREQ step be appended to Gender.html, which contains the results from the preceding PROC MEANS step. The NOTOP suboption causes ODS to write out the required bottom matter tags to Gender.html and to omit writing the top matter tags.

Without the FILENAME statement and the MOD option, the output from the second run would overwrite the results from the first run.

This example assumes that you can map a drive from Windows to point to the same location where the UNIX results are stored. You could also move the file from one server to the other between runs.

```
    filename gender 'n:\trial1\results\Gender.html' mod;
ODS HTML BODY=gender (NOTOP);
proc freq data=clinical;
  tables tx*gender / nocol norow nopct;
      format tx txfmt.;
```

```
run;
ODS HTML CLOSE;
```

The results are shown in Figure 3.51.

Figure 3.50

The MEANS Procedure

Gender	N Obs	Variable	Label	Mean	Std Dev
Female	108	chgsbp	Change in Systolic BP	-25.29	22.28
		chgdbp	Change in Diastolic BP	-20.12	21.82
Male	112	chgsbp	Change in Systolic BP	-26.67	22.79
		chgdbp	Change in Diastolic BP	-16.79	20.16

Figure 3.51

The MEANS Procedure

Gender	N Obs	Variable	Label	Mean	Std Dev
Female	108	chgsbp	Change in Systolic BP	-25.29	22.28
		chgdbp	Change in Diastolic BP	-20.12	21.82
Male	112	chgsbp	Change in Systolic BP	-26.67	22.79
		chgdbp	Change in Diastolic BP	-16.79	20.16

The FREQ Procedure

Frequency

Table of tx by gender

tx(Treatment Group)	gender(Gender)		
	Female	Male	Total
Control	41	35	76
Low Dose	25	40	65
High Dose	42	37	79
Total	108	112	220

Appending to an Existing Table of Contents

In the previous example, we created an HTML file in two steps. That example created only a body file. It is also possible to create a Web page with a table of contents using the same technique. There are just a couple of modifications to add to the previous program. First, we need to add specifications for a CONTENTS file and a FRAME file to the ODS HTML statements.

The FRAME= specification includes the ANCHOR= option. Since the body file is being created in two steps, a different starting value for the anchor tags in each run is required, or two "IDX" anchors will end up in the results. The following example code uses "FIRSTRUN" as the starting value for the anchor tags in the top part of the results.

As in the previous example, the first program is run under UNIX and the second is run under Windows.

```
      ODS HTML BODY='/tmp/trial1/results/GenderBody.html' (NOBOT)
       CONTENTS='/tmp/trial1/results/GenderTOC.html' (NOBOT)
       FRAME='/tmp/trial1/results/GenderFrame.html'
                 ANCHOR='firstrun';
proc means data=clinical mean std maxdec=2;
  class gender;
  var chgsbp chgdbp;
run;
ODS HTML CLOSE;
```

This program creates a frame file, a body file, and a table of contents file with all of the required top matter tags, along with your first set of procedure results. The NOBOT suboption omits the bottom matter tags from the body and contents files. Figure 3.52 shows the resulting HTML output.

The second program is run under Windows, and its output is appended to the files created by the first program under UNIX. The NOTOP suboption is added to the specifications for the BODY and CONTENTS files, thus omitting the top matter tags from these files. The bottom matter tags are inserted by the second program.

This second run requires two FILENAME statements, one for the BODY file and one for the CONTENTS file. Both statements include the MOD option so that results from this second program are appended to the files created by the first program. The ANCHOR= suboption on the FRAME= option specifies as its starting value "SECONDRUN," which is different from the starting value in the first program. The frame file is created as before. It's actually the same between the two runs, but it needs to be referenced each time so that the contents page is built correctly.

```
      filename genderb 'n:\trial1\results\GenderBody.html' mod;
filename gendert 'n:\trial1\results\GenderTOC.html' mod;
ODS HTML BODY=genderb (NOTOP)  CONTENTS=gendert (NOTOP)
       FRAME='n:\trial1\results\GenderFrame.html'
            ANCHOR='secondrun';
```

```
proc freq data=clinical;
   tables tx*gender / nocol norow nopct;
         format tx txfmt.;
run;
ODS HTML CLOSE;
```

Figure 3.53 presents the results from the second program. Notice that the new results have been added to both the body page and the contents page.

Figure 3.52

Table of Contents

1. The Means Procedure
   - Summary statistics

The MEANS Procedure

Gender	N Obs	Variable	Label	Mean	Std Dev
Female	108	chgsbp	Change in Systolic BP	-25.29	22.28
		chgdbp	Change in Diastolic BP	-20.12	21.82
Male	112	chgsbp	Change in Systolic BP	-26.67	22.79
		chgdbp	Change in Diastolic BP	-16.79	20.16

Figure 3.53

Table of Contents

1. The Means Procedure
   - Summary statistics
2. The Freq Procedure
   - Table tx * gender
     - Cross-Tabular Freq Table

The MEANS Procedure

Gender	N Obs	Variable	Label	Mean	Std Dev
Female	108	chgsbp	Change in Systolic BP	-25.29	22.28
		chgdbp	Change in Diastolic BP	-20.12	21.82
Male	112	chgsbp	Change in Systolic BP	-26.67	22.79
		chgdbp	Change in Diastolic BP	-16.79	20.16

The FREQ Procedure

Frequency

Table of tx by gender

	gender(Gender)		
tx(Treatment Group)	Female	Male	Total
Control	41	35	76
Low Dose	25	40	65
High Dose	42	37	79
Total	108	112	220

Moving ODS Output into Your Internet or Intranet Site

If you intend to add your ODS HTML results to an existing Internet or intranet Web site, there are some things you can do to make this simpler. First, you can make your Web pages more portable.

For example, if you created a Web page with the following code, you would end up with Web pages with the file path C:\webfiles embedded in each of the table of contents links.

```
ODS HTML BODY='c:\webfiles\outliers.html'
  CONTENTS='c:\webfiles\toc.html'
  FRAME='c:\webfiles\frame.html';
ODS HTML SELECT EXTREMEOBS;
proc univariate data=clinical;
   var basesbp postsbp chgsbp;
run;
ODS HTML CLOSE;
```

That works fine if you always intend to keep the files in the C:\webfiles directory. However, if you move them to your Web site, you have a problem.

The easiest way to make your files portable so you can post them to a Web server is to use the URL=NONE option. This tells ODS to not embed the absolute path in the links. Instead of pointing to the body page at C:\webfiles\outliers.html, the links will be to outliers.html. This means that if you move all three files together, the links will continue to work correctly.

```
ODS HTML BODY='c:\webfiles\outliers.html'
  CONTENTS='c:\webfiles\toc.html'
  FRAME='c:\webfiles\frame.html' (URL=NONE);
```

Now you can move your files without any problems. Figure 3.54 shows that the frame page now works correctly, even after it has been moved to the Web server at www.myweb.com. The URL=NONE option should be used whenever there is a possibility that you will need to later move the files.

Figure 3.54

SAS Output Frame - Windows Internet Explorer

http://www.myweb.com/frame.html

SAS Output Frame

Table of Contents

1. The Univariate Procedure
   - basesbp
     - Extreme Observations
   - postsbp
     - Extreme Observations
   - chgsbp
     - Extreme Observations

The UNIVARIATE Procedure
Variable: basesbp (Baseline Systolic BP)

Extreme Observations			
Lowest		Highest	
Value	Obs	Value	Obs
100	109	177	13
100	79	177	60
101	90	178	176
101	42	179	96
101	31	179	185

The UNIVARIATE Procedure
Variable: postsbp (Post-Treatment Systolic BP)

Extreme Observations	
Lowest	Highest

Sending HTML Results Automatically via E-Mail

Sometimes you need to e-mail your ODS HTML results to supervising managers or other interested parties. There are several ways to initiate e-mail using ODS. One method involves sending the output directly to the EMAIL engine, sending only the output with no explanatory message. On the other hand, it might be a requirement to send an explanatory note with the results output. Imagine that you had PROC UNIVARIATE syntax to create a quick report on blood pressure treatment group outliers (Extreme Observations):

```
ODS HTML FILE='c:\temp\ClinResults.html' STYLE=sasweb;
** PROC UNIVARIATE code and TITLE statements;
ODS HTML CLOSE;
```

Now that the file has been created with ODS, you next need to use the EMAIL engine in the FILENAME statement to specify the TO, FROM, CC, SUBJECT, and ATTACH information. (Note that false e-mail addresses are shown in the following code, and although real e-mail addresses were used to get the screen shots, the working e-mail addresses are not shown in the Outlook screen shot.)

```
FILENAME myemail EMAIL
        to='ABChang@mycompany.com'
        from='DEFitch@mycompany.com'
        cc='GHIsaacson@mycompany.com'
        subject='Attached Case Report'
        attach='c:\temp\ClinResults.html';
```

Finally, you use a DATA step with a FILE statement that references the fileref for the e-mail. In the e-mail message, you include a reminder about when case reviews are due by writing the message with PUT statements. Depending on the value of your e-mail-related system options, your default e-mail server is launched to send the e-mail and the attachment.

```
data _null_;
  file myemail;
  put 'Remember to review your cases by Friday.';
  put 'Outliers are shown in the attached report.';
  put 'This is a test with an attachment';
  put 'that was created by ODS HTML.';
run;
```

On Windows, with Microsoft Outlook as the default mail server, you might receive the prompt shown in Figure 3.55. When the recipient opens the e-mail file (Figure 3.56), the HTML attachment can be opened and viewed as shown in Figure 3.57.

On operating systems other than Windows, you might need to enlist the help of your mail server administrator (if mail server settings need to change) or your SAS administrator, because some of the e-mail-related options can be changed or specified only when SAS is invoked. For more information, refer to online information on the SAS Web site.

Figure 3.55

Figure 3.56

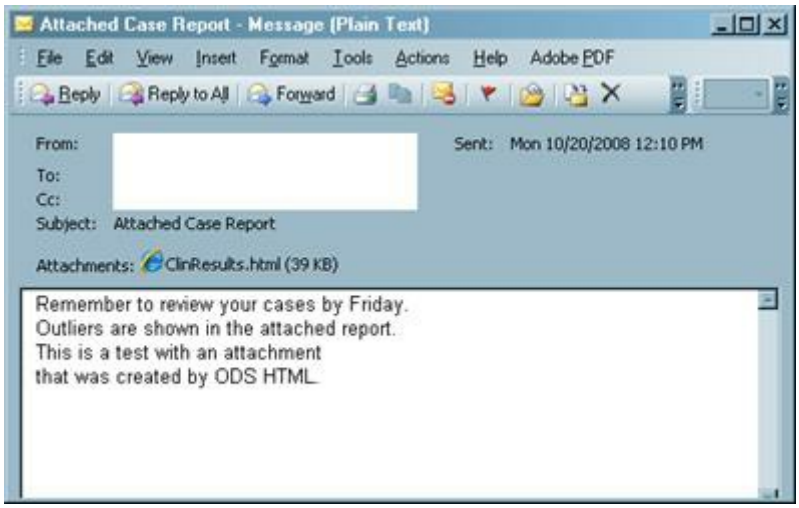


Figure 3.57: Outliers Report Only

Outliers in Treatment Groups

Variable: chgdbp (Change in Diastolic BP)
tx = Control

Extreme Observations			
Lowest		Highest	
Value	Obs	Value	Obs
-55	132	21	7
-50	131	23	5
-49	123	24	26
-42	130	26	2
-42	41	34	38

Importing Your ODS Output into a Spreadsheet

Because the Internet has become so pervasive, HTML is a widely accepted file format. This means that if you create ODS HTML output from your SAS program, you can probably read it into any other software you care to use. For example, you can take your HTML output and import it into a spreadsheet program such as Microsoft Excel. Since ODS HTML output is largely composed of tables, with rows and columns, a spreadsheet can easily interpret the results and format them properly.

The following code analyzes the CHGDBP and CHGSBP variables in the data set CLINICAL. It saves the same report three different ways that a spreadsheet program can easily open. ODS gives you three ways to generate formatted output for a spreadsheet program: 1) use HTML 3.2 or HTML 4.0 tags to create an HTML file; 2) use Microsoft HTML tags to create an HTML file; or 3) use Microsoft Spreadsheet Markup Language to create an XML file (SpreadsheetML). The following code shows these three choices.

```
ODS HTML BODY='avgbp_html4.xls' style=sasweb;
ODS MSOFFICE2K BODY='avgbp_msoffice.xls' style=sasweb;
ODS TAGSETS.EXCELXP BODY='avgbp_xp.xls' style=sasweb;
title 'Average BP Statistics';
proc report data=clinical nowd split=' ' style(header)={vjust=b};
  column tx ('Average' ChgDBP ChgSBP);
  define tx / group "Group" format=txft.;
  define ChgDBP / analysis mean format=10.1;
  define ChgSBP / analysis mean format=10.1;
 run;
ODS _ALL_ CLOSE;
```

To import the files into a spreadsheet, simply invoke the spreadsheet program, and open each file directly. Figure 3.58 shows what you get if you open the Excel file created with the ODS HTML statement. This spreadsheet looks almost exactly like the version created with the ODS MSOFFICE2K statement.

This Excel spreadsheet can now be easily edited or saved as an Excel XLS-type file. After the HTML table has been converted to Excel rows and columns, you can use Excel to add new computations. You can also add your results to an existing spreadsheet. This is helpful if you're combining results from a number of sources. You can import your HTML or XML results into a spreadsheet by cutting and pasting between the browser window and the spreadsheet. Starting with Microsoft Office 2002, Excel can read Spreadsheet Markup Language XML files. Because the EXCELXP tagset produces Spreadsheet Markup Language that is compliant with Microsoft Office 2002/2003 standards, the resulting XML file must be opened with Excel 2002 or later.

When you use this XML technique, your spreadsheet ends up looking something like Figure 3.59. The table itself retains the colors and styles of the HTML version. However, because the table row and column widths might not be adjusted properly to fit your SAS output, you might find that you have to adjust column widths. In addition, the EXCELXP tagset provides an option to adjust the cell width of the columns, as well as a variety of other

options that enable you to alter Excel sheet properties, such as print orientation, sheet name, and auto-filtering.

Figure 3.58

HTML 4.0

avgbp_html4.xls

	A	B	C
1	Average BP Statistics		
2			
3		Average	
4	Group	Change in Diastolic BP	Change in Systolic BP
5			
6			
7			
8	Control	-15.4	-27.3
9	High Dose	-24.5	-26.6
10	Low Dose	-14.6	-23.8
11			

Microsoft HTML

avgbp_msoffice.xls

	A	B	C	D	E	F	G
1	Average BP Statistics						
2							
3		Average					
4		Change in	Change in				
5		Diastolic	Systolic				
6		BP	BP				
7	Group						
8	Control	-15.4	-27.3				
9	High Dose	-24.5	-26.6				
10	Low Dose	-14.6	-23.8				

Figure 3.59: SpreadsheetML Created by EXCELXP Tagset

avgbp_tagsetsxp.xls

	A	B	C
1		Average	
2	Group	Change in Diastolic BP	Change in Systolic BP
3	Control	-15.4	-27.3
4	High Dose	-24.5	-26.6
5	Low Dose	-14.6	-23.8
6			

Another way to move your SAS output to Excel is to use the CSV output destination. The syntax is ODS CSV FILE='myfile.csv'; (followed by ODS CSV CLOSE;). If you want your SAS titles, procedure titles and by line information in the CSV output files, you only need to change your destination from CSV to CSVALL. Although the output from these destinations does not have any color or font formatting, they represent another alternative to using PROC EXPORT or the EXPORT Wizard.

Using ODS Suboptions

ODS HTML suboptions perform different actions. Table 3.1 lists several of them along with the names of examples where they are featured in this chapter.

Table 3.1: Selected ODS HTML Suboptions and Related Examples

Suboption	Behavior	Featured in
(TITLE=)	Inserts a <TITLE> tag into the HTML file being created.	"Changing the Title Bar"
(ID= *identifier*) or (*identifier*)	Allows multiple instances of the same ODS destination to be open at the same time.	"Creating Multiple Copies of the Same Output"
		"Creating and Using Cascading Style Sheets"
(NO_TOP_MATTER) or (NOTOP)	Suppresses everything between the <DOCTYPE> tag and the <BODY> tag that normally goes at the top of the HTML file. Note that this suboption removes any reference to a CSS style.	"Appending to an Existing HTML File"
		"Appending to an Existing Table of Contents"
(NO_BOTTOM_MATTER) or (NOBOT)	Suppresses the </HTML> tag that normally goes at the bottom of the HTML file.	"Appending to an Existing HTML File"
		"Appending to an Existing Table of Contents"
(URL=)	Specifies a URL for any internal hyperlinks that ODS might need to build for anchor tags, such as those used for <IMG> tags, <A> tags, or <LINK> tags. ODS uses this URL instead of the default URL (usually the filename). Note that this suboption is most frequently used in the PATH=, the GPATH=, or the STYLESHEET= option.	"Exploring the HTML Body File: Graphics"

Chapter 4
RTF Output

HTML files on the Web are one common way to share your results with other users. However, there's another piece of software your users have in common: the word processor.

Virtually all word processors can import Rich Text Format (RTF) files. And ODS can create several types of RTF output. Starting in SAS 9.2, ODS can create a tagset template version of RTF output: TAGSETS.RTF.

RTF output has all of the colors and formatting you've just seen in the previous chapter. Moreover, in some ways, RTF output is even better. Word processors give more control of the structure and appearance of tables than HTML, and ODS takes full advantage of this. In addition, editing ODS HTML files after their creation is not encouraged because of the complexity of the HTML syntax. However, editing of the RTF files with a word processor is expected.

This syntax for producing ODS RTF output is similar to the syntax for ODS HTML. However, ODS has additional features, options, and styles that make RTF a powerful destination in its own right. Traditional RTF output is covered in the first part of this chapter, and specific examples of TAGSETS.RTF are shown in the last part of the chapter.

This chapter uses the data sets CHEMTEST, CLIPS, and HR to illustrate the concepts. These data sets are available on the author pages for this book at http://support.sas.com/authors.

Creating Your First RTF File

The best thing about creating RTF output is that you have to learn only one new piece of syntax: just change the keyword HTML to RTF in your ODS statement. It takes only a single simple command to begin creating an RTF file:

```
ODS RTF FILE='myfile.rtf';
```

One simple change that you can make for RTF output is to use the STYLE= option in the invocation statement to generate a report with a completely different look and feel for the output.

```
ODS RTF FILE='myfile2.rtf' STYLE=JOURNAL;
```

Just as with the ODS HTML statement, you can list a literal filename, or use a fileref to refer to a previously defined filename. Next, you insert your procedure or DATA step to create some output. Finally, you close the RTF file with the following:

```
ODS RTF CLOSE;
```

To show how this works, we'll create two simple reports using PROC MEANS. For the first report, we'll use the default invocation. Then we'll create a second file using the JOURNAL style. Here is the code for the first report:

```
ODS RTF FILE='Payroll.rtf';
title 'Payroll Report';
proc means data=hr nonobs maxdec=0 mean sum;
  class Department;
  var AnnualSalary;
run;
ODS RTF CLOSE;
```

For the second file, the only change to the above program is to add the STYLE= option to the ODS RTF statement:

```
ODS RTF FILE='Payroll.rtf' STYLE=JOURNAL;
```

The resulting RTF files can be opened using your favorite word processor. Figure 4.1 shows what the two different styles look like if you open the output files with Microsoft Word. The RTF output looks similar to the HTML output, but the report table is more refined. The table headers are either shaded or bold, depending on the style used. Chapter 14, "Understanding the Built-In Style Template Definitions," covers how to change the style definition of your output. You could use either the default RTF style or the JOURNAL style as a place to start.

After the output is opened in a word processor, you can manipulate the output just like any other document. You can change the font face or size, use different colors or borders for the table, apply heading formats to the titles, or add explanatory text. In Word, you can also use the Table AutoFormat tool to create more elaborate table designs.

How is all this accomplished? The RTF file contains not just the information in your results, but also a variety of complex formatting information. Figure 4.2 shows a bit of the

RTF file for this example, to give you an idea of how the files look internally. Unlike the HTML files in the previous chapter, this file is not easily understood (except by word processors). For that reason, we will not go into the details of RTF files.

Figure 4.1

Default RTF Style

Payroll Report

The MEANS Procedure

Analysis Variable : AnnualSalary Annual Salary		
Department	**Mean**	**Sum**
Administration	51014	3571000
Manufacturing	51507	10456000
Marketing	53465	3849500
R&D	53587	2786500

JOURNAL Style

Payroll Report

The MEANS Procedure

Analysis Variable : AnnualSalary Annual Salary		
Department	*Mean*	*Sum*
Administration	51014	3571000
Manufacturing	51507	10456000
Marketing	53465	3849500
R&D	53587	2786500

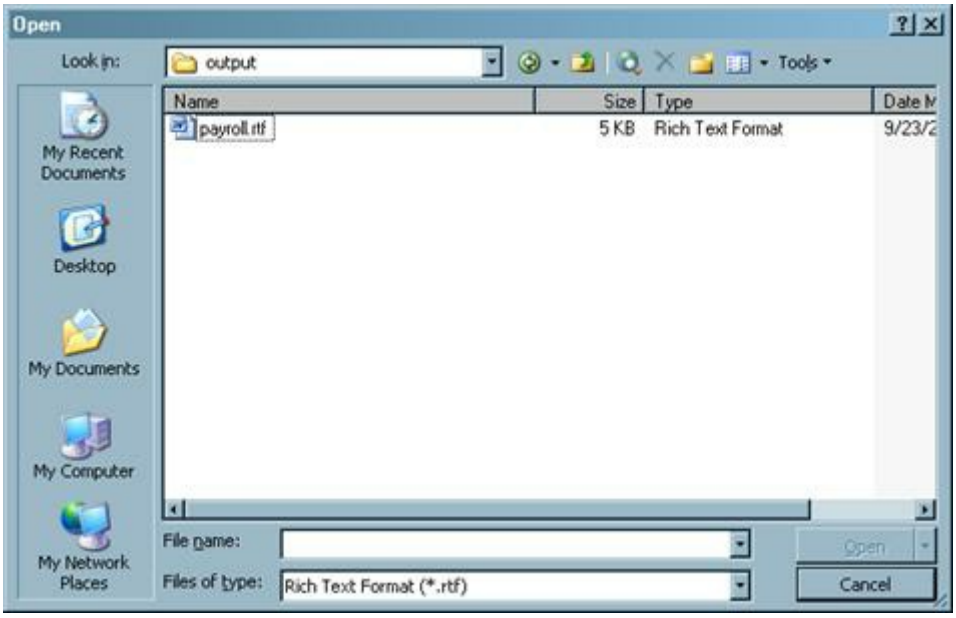
Figure 4.2

{\rtf1\ansi\ansicpg1252\uc1\deff0\deflang1033\deflangfe1033
{\fonttbl
{\f1\froman\fprq2\fcharset0 Times;}
}
{\colortbl;\red0\green0\blue0;\red0\green0\blue255;\red0\green255\blue255;\red0\green255\blue0;\red255\green0\blue255;\red2
\red0\green0\blue128;\red0\green128\blue128;\red0\green128\blue0;\red128\green0\blue128;\red128\green0\blue0;\red128\gree
}{\stylesheet{\widctlpar\adjustright\fs20\cgrid\snext0 Normal;}{*\cs10\additive Default Paragraph Font;}
}{\info{\title V8 SAS System Output}{\author SAS Version 8}{\operator SAS Version 8}{\version1}
{\creatim\yr2000\mo6\dy27\hr7\min23\sec37}}
\widowctrl\ftnbj\aenddoc\formshade\viewkind1\viewscale75\pgbrdrhead\pgbrdrfoot\fet0\paperw11519\paperh15023\margl360\r
\sectd\linex0\endnhere\headery720\footery720\marglsxn360\margrsxn360\margtsxn360\margbsxn360

(Remaining RTF code trimmed to save space.)

Importing into a Word Processor

There's not much to importing your file into a word processor. If you're on a Windows platform, you can just double-click the file and it should open up in your default word processor. Alternatively, you can open the word processor first, and then open the RTF file from there.



If you are using Microsoft Word 2003, select **File** and then **Open** from the menu. This brings up a dialog box where you can specify RTF and the name of your file. The dialog box is shown above, and the resulting Word document is shown on the previous page in Figure 4.1.

If you are using WordPerfect, OpenOffice, or StarOffice, the process is the same. You select **File** and then **Open**, choose RTF as your file type, and then type the name of the file. Figure 4.3 shows how the RTF file looks in WordPerfect. Figure 4.4 shows the same file in OpenOffice. Keep in mind that even though you might be able to *open* ODS-created RTF files in other word processing applications, SAS R&D exclusively tests ODS RTF files in Microsoft Word. Therefore, only Microsoft Word rendering issues are considered as possible problems or defects. The recommendation from SAS Technical Support is that you use Microsoft Word, if possible, to open and render RTF files created by ODS.

ODS RTF output is designed to be compatible with Word 2003 and later.[1] You can import RTF output into an earlier word processor, but results can vary. If you're on a platform other than Windows, you might need to adjust some option settings when you create the RTF file, and you might need to copy the file over to another computer in order to use your word processor. See the Appendix, "Operating System Differences," for information about other operating systems.

Figure 4.3: RTF Output Imported into WordPerfect

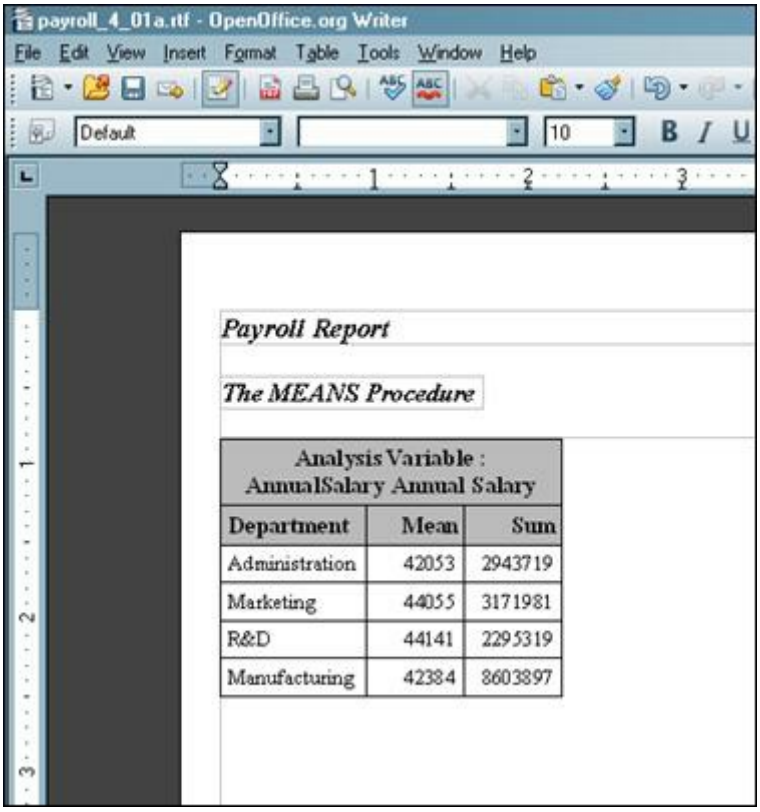
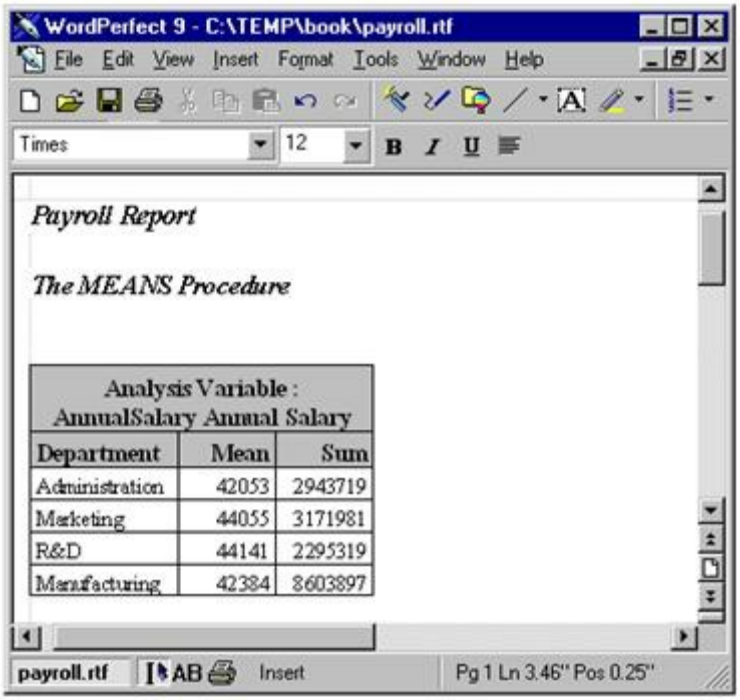


Figure 4.4: RTF Output Imported into OpenOffice



Working with Headers and Footers in Microsoft Word

The previous samples of output from Microsoft Word showed the document in print preview mode. This was done so that you could see how the titles would appear in a printed Word document. However, this is not what you see when you first open an ODS RTF file in Word.

When you first open an RTF document, you see it in normal view or page layout view (depending on your default settings). In this view, you might be surprised by how faint the titles and footnotes look. Instead of the bold fonts you were expecting, you see pale gray text.

The RTF file from our previous examples is shown in Figure 4.5, where you can see that the titles are barely visible. This is because Word takes your titles and footnotes and turns them into page headings and footers, which are displayed in a grayed-out font on the screen. To see them properly, you have to do a print preview, or switch to headers and footers view, as shown in Figure 4.6.

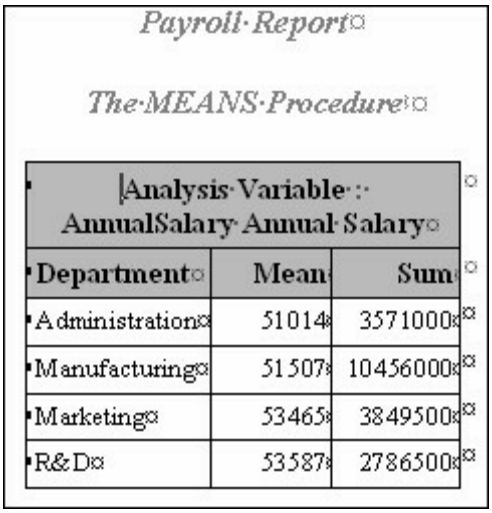
Because ODS creates RTF files that specify your titles and footnotes as headers and footers, they get converted to page headers and page footers in most word processors, and might not look correct on the screen. But rest assured, your document will print correctly.

If you would prefer that the titles and footnotes not be put in the header and footer, you can use the BODYTITLE option to force them into the body of the document. Here is the syntax:

ODS RTF FILE='*filename*' BODYTITLE;

Figure 4.7 shows the resulting RTF file. The only difference between the results shown in Figure 4.7 and the previous version shown in Figure 4.5 is that now the two headings are part of the main document and can be edited without switching to a different view.

Figure 4.5: RTF Output with Title in Header



Payroll Report

The MEANS Procedure

Analysis Variable : AnnualSalary Annual Salary		
Department	Mean	Sum
Administration	51014	3571000
Manufacturing	51507	10456000
Marketing	53465	3849500
R&D	53587	2786500

Figure 4.6

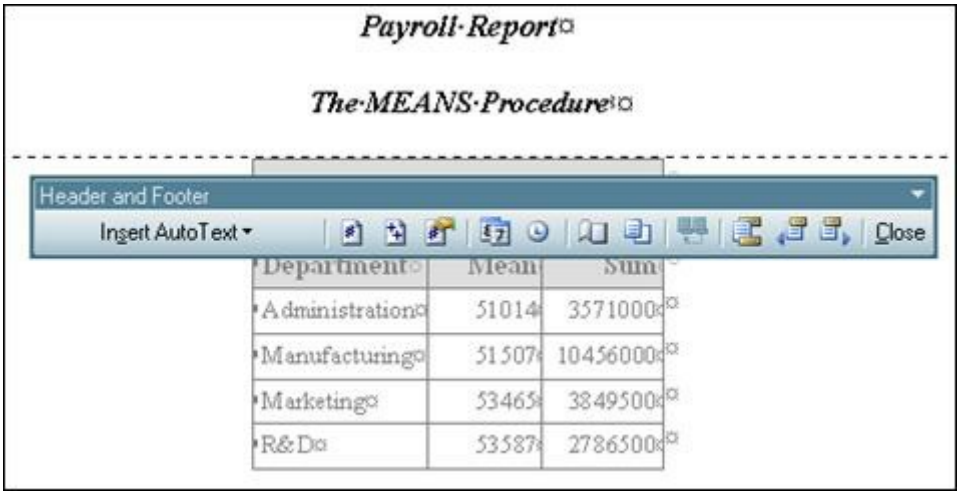


Figure 4.7: RTF Output with Title in Body

Handling Page Breaks in RTF Output

In order for ODS to figure out how to fit your SAS output onto a word processor page, ODS uses the standard PAPERSIZE of LETTER, a page size of 8½ by 11 inches. Then, ODS uses this page width to determine how much information will fit in each *row* of your table. This ensures that your tables will not be too wide for the page. If a table will not fit the page width, ODS breaks up the table and wraps the output.

For example, the following code produces the output shown in Figure 4.8.

```
      ODS RTF FILE='SalaryWide.rtf';
title 'Salary Report';
proc freq data=hr;
  format AnnualSalary salft.;
  tables Department*AnnualSalary / norow nocol nopct;
run;
ODS RTF CLOSE;
```

Notice how this frequency table is broken into smaller tables so that it will fit in the page width. The first two tables are shown. Actually, it took four smaller tables to hold all of the salary ranges. ODS makes sure that your output will fit on a standard page width.

However, ODS does not control the *length* of your output when creating traditional RTF files. ODS hands off this responsibility to your word processor. If a table is too long to fit on the page, it is up to your word processor to break the table up and to repeat the column headings on the next page. The following code illustrates how this works. The FREQ procedure produces a tall table that does not fit on a single page. Figure 4.9 shows how Microsoft Word has handled the table, breaking it into two parts and repeating the headings.

```
      ODS RTF FILE='SalaryTall.rtf';
title 'Salary Report';
proc freq data=hr;
  tables AnnualSalary / nopct nocum;
run;
ODS RTF CLOSE;
```

This traditional RTF behavior assures that headers and footers are taken into account when the decision is made about where to break up the long tables in your output. However, because ODS does not control page breaks in traditional RTF output, this means that procedures that produce page totals might not put them at the bottom of a physical page. Also, the TABULATE procedure does not place a “Continued” label at the bottom of a page when the table is split onto two pages.

Why does ODS handle output width and height differently? If ODS used the standard page height, but you added a header or edited your text to change the font or insert a note in your Word document, your output would no longer fit on the page, and might break awkwardly. If you want to produce output that is measured in both height and width, then the new measured RTF tagset in SAS 9.2 provides you more control over vertical page measurement. See the

end of this chapter for more examples of the new TAGSETS.RTF destination.

Figure 4.8

Salary Report

The FREQ Procedure

Table of Department by AnnualSalary					
Department(Department)	AnnualSalary(Annual Salary)				
Frequency	<$40,000	$40,000-$44,999	$45,000-$49,999	$50,000-$54,999	$55,000-$59,999
Administration	38	6	4	3	3
Manufacturing	108	19	11	5	3
Marketing	43	2	3	2	2
R&D	30	1	5	1	2
Total	219	28	23	11	12

Table of Department by AnnualSalary					
Department(Department)	AnnualSalary(Annual Salary)				
Frequency	$60,000-$64,999	$65,000-$69,999	$70,000-$74,999	$75,000-$79,999	$80,000-$84,999
Administration	1	1	3	2	2
Manufacturing	6	8	6	3	7
Marketing	1	2	1	0	3
R&D	2	1	0	1	0
Total	10	12	10	6	12

Figure 4.9

The top of the first page

Salary Report

The FREQ Procedure

Annual Salary	
AnnualSalary	Frequency
$35,000	191
$35,500	2
$36,000	6
$36,500	3

The bottom of the second page

$48,500	1
$49,000	3
$49,500	2
$50,000	1
$50,500	1
$51,000	2
$52,000	2

Controlling Page Orientation

When you have a table that is tall and skinny, it makes sense to print it in portrait layout. But when you have a table that is wider than it is tall, a landscape layout makes more sense. By default, if you don't change any options, ODS RTF creates portrait output.

If you have a wide table, like that shown in Figure 4.10, you might decide that you don't like the default portrait orientation. In this PROC MEANS analysis of the data set CLIPS, the variable labels for variables MENTIONS and COMPETE are too wide to fit their columns, so the text wraps and does not align with the statistics. This can be remedied by switching the orientation of the output to landscape. Here is the code that produces Figure 4.10:

```
ODS RTF FILE='CLIPSStats.rtf';
proc means data=clips n mean std min p25 median p75 max nway
        maxdec=1;
  class tone;
  var mentions compete;
run;
ODS RTF CLOSE;
```

The syntax for changing the orientation is not part of ODS. Instead, you use the SAS system option ORIENTATION=. For example, the following statement placed before the ODF RTF statement above changes the orientation of the output to landscape and produces the output in Figure 4.11. Notice that the table is not much wider, but now there's room for the variable labels in their column.

```
OPTIONS ORIENTATION=LANDSCAPE;
```

To switch the orientation back to the default, just issue the following command:

```
OPTIONS ORIENTATION=PORTRAIT;
```

Using this option makes it fairly simple to switch back and forth between orientations. It is also possible to switch orientations within a single RTF file. To do this, add an extra ODS RTF statement after the OPTIONS ORIENTATION= statement that changes the orientation. This tells ODS to check the options setting again before generating the next output. For example, the following program creates two tables, the first with a landscape layout and the second with a portrait layout:

```
OPTIONS ORIENTATION=LANDSCAPE;
ODS RTF FILE='BothOrientations.rtf';
proc print data=clips(obs=5);
run;
OPTIONS ORIENTATION=PORTRAIT;
ODS RTF;
proc means data=clips n mean std min max maxdec=1;
run;
ODS RTF CLOSE;
```

The resulting file is shown in Figure 4.12. The PROC PRINT output is landscape, and then the orientation changes to portrait for the second page that displays the PROC MEANS output.

If you run this code without including the extra ODS RTF statement after the change of orientation, you will get an RTF file that is only in landscape orientation. Any subsequent changes in orientation are ignored unless the OPTIONS ORIENTATION= statement is followed by another ODS RTF statement.

Figure 4.10

The MEANS Procedure

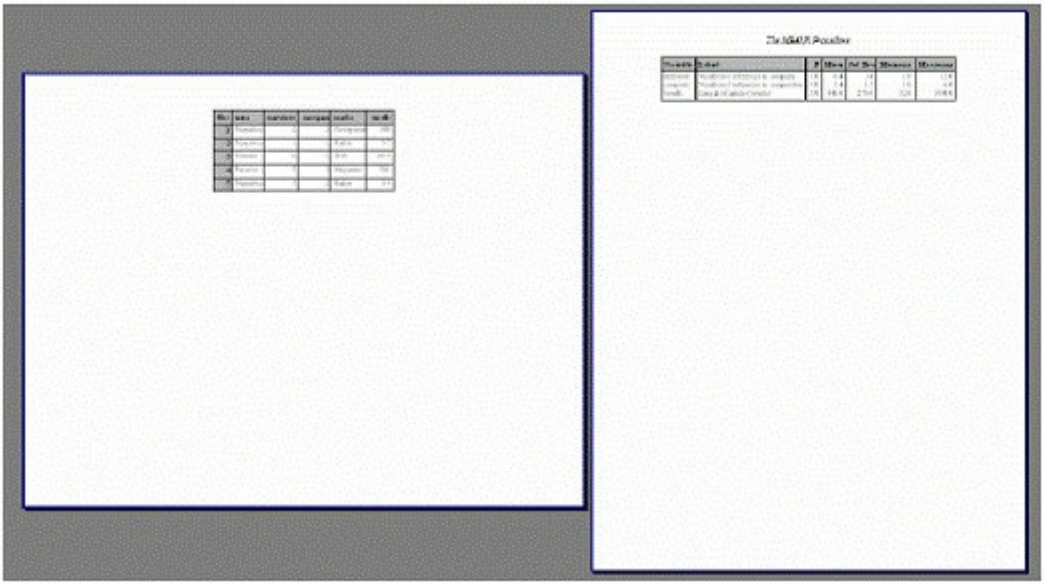
Tone of content	N Obs	Variable	Label	N	Mean	Std Dev	Minimum	25th Pctl	Median	75th Pctl	Maximum
Negative	105	mentions	Number of references to company	105	5.8	3.5	1.0	3.0	5.0	8.0	12.0
		compete	Number of references to competition	105	3.8	1.7	1.0	2.0	4.0	5.0	6.0
Neutral	111	mentions	Number of references to company	111	6.2	3.7	1.0	3.0	6.0	10.0	12.0
		compete	Number of references to competition	111	3.2	1.7	1.0	2.0	3.0	5.0	6.0
Positive	114	mentions	Number of references to company	114	7.0	3.0	1.0	5.0	7.0	9.0	12.0
		compete	Number of references to competition	114	3.2	1.7	1.0	2.0	3.0	4.0	6.0

Figure 4.11

The MEANS Procedure

Tone of content	N Obs	Variable	Label	N	Mean	Std Dev	Minimum	25th Pctl	Median	75th Pctl	Maximum
Negative	105	mentions	Number of references to company	105	5.8	3.5	1.0	3.0	5.0	8.0	12.0
		compete	Number of references to competition	105	3.8	1.7	1.0	2.0	4.0	5.0	6.0
Neutral	111	mentions	Number of references to company	111	6.2	3.7	1.0	3.0	6.0	10.0	12.0
		compete	Number of references to competition	111	3.2	1.7	1.0	2.0	3.0	5.0	6.0
Positive	114	mentions	Number of references to company	114	7.0	3.0	1.0	5.0	7.0	9.0	12.0
		compete	Number of references to competition	114	3.2	1.7	1.0	2.0	3.0	4.0	6.0

Figure 4.12



You cannot change the orientation using the LINESIZE= and PAGESIZE= options. These options can be used to change the orientation of listing output, but they have no effect on most RTF output. This is also true of HTML and Printer output.

Controlling Page Size

Although you can't use the SAS system option PAGESIZE= to control page size in RTF output, you do have some control. You can use the SAS system option PAPERSIZE= to choose a different page size for your output.

For example, if your output is so wide that even using ORIENTATION=LANDSCAPE is not sufficient to fit your results to one page, one option is to change to wider paper. For example, you can switch to a page size appropriate for legal-size paper. When you open up your document in a word processor, it will be set up for printing on legal-size paper. The syntax for requesting a different paper size is simple:

```
OPTIONS PAPERSIZE=LEGAL;
```

The default setting for this option is PAPERSIZE=LETTER. Other options vary depending on your operating system. On Windows, there are dozens of options. Some of the more common choices are Tabloid (11x17), Legal, and Letter.

To determine the paper sizes available to ODS, you need to view the SAS Registry. To view the SAS Registry from your SAS session, first open the Registry Editor by selecting **Accessories u Solutions u Registry Editor**. From the Registry Editor window, select **CORE u PRINTING u PAPER SIZES**. When you select **PAPER SIZES**, SAS displays a list of available paper sizes.

Another way to view this list is to issue the following code, which lists in the SAS log the paper size names along with their associated dimensions.

```
proc registry list startat="CORE\PRINTING\PAPER SIZES";
run;
```

For example, in SAS 9.2, the SAS log would show you these paper sizes for Legal and Tabloid:

[Legal] Height=double:14 Units="IN" Width=double:8.5	[Tabloid] Height=double:17 Units="IN" Width=double:11

Figure 4.13 shows an example of a table that is too wide for even landscape orientation using the default paper size. The output wraps onto page 2. To fit this output on one page, the following code uses not only landscape orientation but also changes the paper size to Legal. Figure 4.14 presents the results.

```
OPTIONS ORIENTATION=LANDSCAPE PAPERSIZE=LEGAL;
ODS RTF FILE='VeryWide.rtf';
proc means data=clips maxdec=1
        mean std min p1 p5 p10 p25 p50 p75 p90 p95 p99 max
        stderr probt clm lclm cv css uss;
  class tone;
  var mentions compete;
```

```
run;
ODS RTF CLOSE;
```

Figure 4.13: Output Wraps Across Two Pages

The MEANS Procedure

Tone of content	N Obs	Variable	Label	Mean	Std Dev	Minimum	1st Pctl	5th Pctl	10th Pctl	25th Pctl	50th Pctl	75th Pctl	90th Pctl	95th Pctl	99th Pctl	Maximum	Std Error	Pr > |t|	Lower 95% CL for Mean
Negative	105	Mentions Compete	Number of references to company Number of references to competition	5.8 3.8	3.5 1.7	1.0 1.0	1.0 1.0	1.0 1.0	1.0 1.0	3.0 2.0	5.0 4.0	8.0 5.0	11.0 6.0	12.0 6.0	12.0 6.0	12.0 6.0	0.3 0.2	<.0001 <.0001	5.1 3.4
Neutral	111	Mentions Compete	Number of references to company Number of references to competition	6.2 3.2	3.7 1.7	1.0 1.0	1.0 1.0	1.0 1.0	1.0 1.0	3.0 2.0	6.0 3.0	10.0 5.0	11.0 6.0	12.0 6.0	12.0 6.0	12.0 6.0	0.3 0.2	<.0001 <.0001	5.5 2.9
Positive	114	Mentions Compete	Number of references to company Number of references to competition	7.0 3.2	3.0 1.7	1.0 1.0	1.0 1.0	2.0 1.0	3.0 1.0	5.0 2.0	7.0 3.0	9.0 4.0	11.0 6.0	12.0 6.0	12.0 6.0	12.0 6.0	0.3 0.2	<.0001 <.0001	6.4 2.9

The MEANS Procedure

Tone of content	N Obs	Variable	Label	Upper 95% CL for Mean	Coeff of Variation	Corrected SS	Uncorrected SS
Negative	105	Mentions Compete	Number of references to company Number of references to competition	6.5 4.1	60.9 44.8	1284.5 293.6	4782.0 1772.0
Neutral	111	Mentions Compete	Number of references to company Number of references to competition	6.9 3.5	59.0 52.7	1485.9 314.2	5800.0 1456.0
Positive	114	Mentions Compete	Number of references to company Number of references to competition	7.6 3.5	43.1 52.5	1033.0 324.1	6633.0 1512.0

Figure 4.14

The MEANS Procedure

Tone of content	N Obs	Variable	Label	Mean	Std Dev	Minimum	1st Pctl	5th Pctl	10th Pctl	25th Pctl	50th Pctl	75th Pctl	90th Pctl	95th Pctl	99th Pctl	Maximum	Std Error	Pr > |t|	Lower 95% CL for Mean	Upper 95% CL for Mean	Coeff of Variation	Corrected SS	Uncorrected SS
Negative	105	Mentions Compete	Number of references to company Number of references to competition	5.8 3.8	3.5 1.7	1.0 1.0	1.0 1.0	1.0 1.0	1.0 1.0	3.0 2.0	5.0 4.0	8.0 5.0	11.0 6.0	12.0 6.0	12.0 6.0	12.0 6.0	0.3 0.2	<.0001 <.0001	5.1 3.4	6.5 4.1	60.9 44.8	1284.5 293.6	4782.0 1772.0
Neutral	111	Mentions Compete	Number of references to company Number of references to competition	6.2 3.2	3.7 1.7	1.0 1.0	1.0 1.0	1.0 1.0	1.0 1.0	3.0 2.0	6.0 3.0	10.0 5.0	11.0 6.0	12.0 6.0	12.0 6.0	12.0 6.0	0.3 0.2	<.0001 <.0001	5.5 2.9	6.9 3.5	59.0 52.7	1485.9 314.2	5800.0 1456.0
Positive	114	Mentions Compete	Number of references to company Number of references to competition	7.0 3.2	3.0 1.7	1.0 1.0	1.0 1.0	2.0 1.0	3.0 1.0	5.0 2.0	7.0 3.0	9.0 4.0	11.0 6.0	12.0 6.0	12.0 6.0	12.0 6.0	0.3 0.2	<.0001 <.0001	6.4 2.9	7.6 3.5	43.1 52.5	1033.0 324.1	6633.0 1512.0

Controlling Page Breaks

When your RTF output produces a number of tables, ODS tries to insert page breaks appropriately. By default, it puts a page break between procedures and between BY groups. This can result in RTF files with a few tables spanning many pages with a lot of white space.

For example, if you analyze the data set CLIPS with a PROC MEANS step followed by a PROC TABULATE step, you can end up with the results shown in Figure 4.15. The entire first page is devoted to the small table produced by the PROC MEANS step. The entire second page is devoted to the first BY group from the TABULATE procedure. These two pages are followed by four more pages, each displaying a single BY group from the PROC TABULATE step.

In HTML output, each of the procedures would just be displayed in order with no wasted space. You can achieve the same effect in RTF output by using the STARTPAGE= option. The syntax is as follows:

```
ODS RTF FILE='filename.rtf' STARTPAGE=NO;
```

This option setting tells ODS to not send any page breaks to the RTF output. The pagination of your results is handled by your word processor. For example, if this option is added to our sample code, the results now fit in the two pages shown in Figure 4.16.

```
ODS RTF FILE='Multipage.rtf' STARTPAGE=NO;
proc means data=clips maxdec=0 nonobs mean std;
  class media;
  var words;
run;
proc tabulate data=clips f=8.;
  class media tone;
  var words;
  by tone;
  table min p1 p5 p10 p25 p50 p75 p90 p95 p99 max,
    words*media;
run;
ODS RTF CLOSE;
```

If you are creating RTF output from SAS/GRAPH procedures (without any tabular output), STARTPAGE=NO is not even necessary. This is because graphic images are converted to an internal RTF embedded format, and graphics get only as much space on the page as they need.

Figure 4.15

The MEANS Procedure

Analysis Variable : words Length of article (words)		
Media	Mean	Std Dev
Magazine	537	299
Newspaper	303	293
Radio	269	142
TV	318	149
Web	562	308

Tone of content=Negative

	Length of article (words)			
	Media			
	Magazine	Newspaper	Radio	TV
Min	60	133	41	32
P1	60	133	41	32
P5	74	170	41	119
P10	90	190	88	133
P25	318	299	163	242
P50	534	486	322	348
P75	693	693	432	433
P90	976	910	469	490
P95	983	1037	496	517
P99	1023	1044	496	523
Max	1023	1044	496	523

Figure 4.16

The MEANS Procedure

Analysis Variable : words Length of article (words)		
Media	Mean	Std Dev
Magazine	537	299
Newspaper	303	293
Radio	269	142
TV	318	149
Web	562	308

Tone of content=Negative

	Length of article (words)			
	Media			
	Magazine	Newspaper	Radio	TV
Min	60	133	41	32
P1	60	133	41	32
P5	74	170	41	119
P10	90	190	88	133
P25	318	299	163	242
P50	534	486	322	348
P75	693	693	432	433
P90	976	910	469	490
P95	983	1037	496	517
P99	1023	1044	496	523
Max	1023	1044	496	523

Tone of content=Neutral

	Length of article (words)				
	Media				
	Magazine	Newspaper	Radio	TV	Web
Min	78	64	40	33	83
P1	78	64	40	33	83
P5	107	68	40	37	93
P10	117	91	67	62	139
P25	203	166	126	191	219

	Length of article (words)				
	Media				
	Magazine	Newspaper	Radio	TV	Web
P50	310	439	224	344	378
P75	611	693	337	433	796
P90	942	868	471	474	847
P95	992	890	470	473	1018
P99	1007	1003	479	523	1039
Max	1007	1003	479	523	1039

Tone of content=Positive

	Length of article (words)				
	Media				
	Magazine	Newspaper	Radio	TV	Web
Min	03	60	31	32	38
P1	03	60	31	32	38
P5	07	73	66	32	69
P10	131	132	65	36	100
P25	206	277	146	82	285
P50	529	484	237	373	563
P75	828	792	346	473	826
P90	952	901	475	479	959
P95	970	968	481	486	1019
P99	1006	997	483	486	1032
Max	1006	997	483	486	1032

Controlling Page Margins

You might have noticed that when you open a word processor document that was created using ODS RTF, the document has very narrow margins. By default, ODS creates RTF output with one-quarter- inch margins on the top, bottom, right, and left sides. You can see these margins by opening an RTF file created by ODS and selecting **Page Setup**. Figure 4.17 shows the margin settings from the Page Setup window in Microsoft Word 2003. All four margins are set to 0.25", which is the default value for ODS RTF output (set in the style template in SAS 9.2).

Starting in SAS 9.2, it's easy to adjust the margins by changing the SAS system options for margins: TOPMARGIN=, BOTTOMMARGIN=, LEFTMARGIN=, and RIGHTMARGIN=. For example, the following statement preceding your ODS RTF statement changes the four margins to one-half inch.

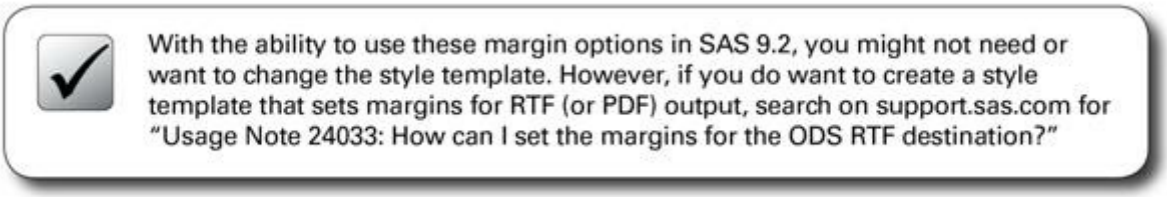
```
OPTIONS TOPMARGIN=.5in BOTTOMMARGIN=.5in LEFTMARGIN=.5in
    RIGHTMARGIN=.5in;
```

When you open your RTF document and look at the document properties, you will see the margin settings as shown in Figure 4.18.

You can also use quotes around the dimension specification. If you issued this OPTIONS statement, you would then see the margin settings shown in Figure 4.19.

```
OPTIONS TOPMARGIN="1.5in" BOTTOMMARGIN="1in" LEFTMARGIN="1in"
    RIGHTMARGIN="1in";
```

In earlier SAS versions, the RTF output used fixed margin settings of one-quarter inch for all four margins. In order to change the margins, you needed to change the style template.



With the ability to use these margin options in SAS 9.2, you might not need or want to change the style template. However, if you do want to create a style template that sets margins for RTF (or PDF) output, search on support.sas.com for "Usage Note 24033: How can I set the margins for the ODS RTF destination?"

Figure 4.17

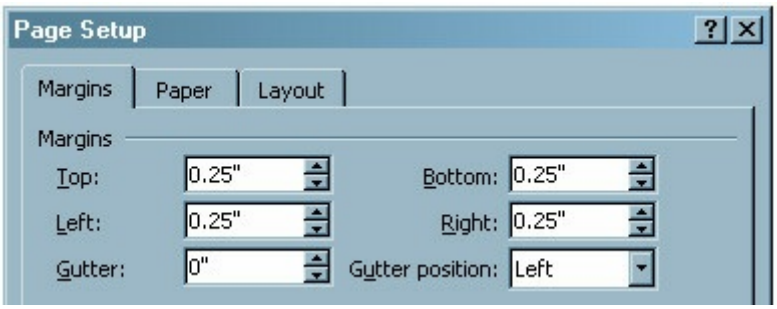


Figure 4.18

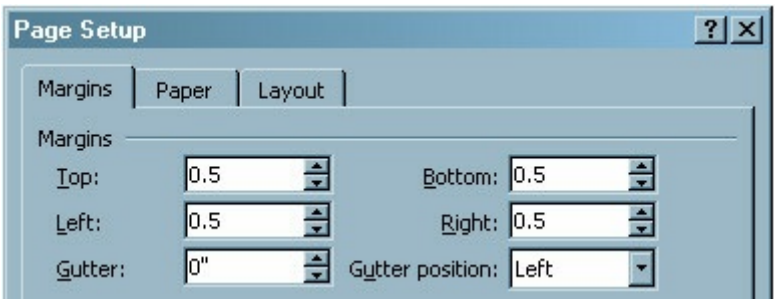
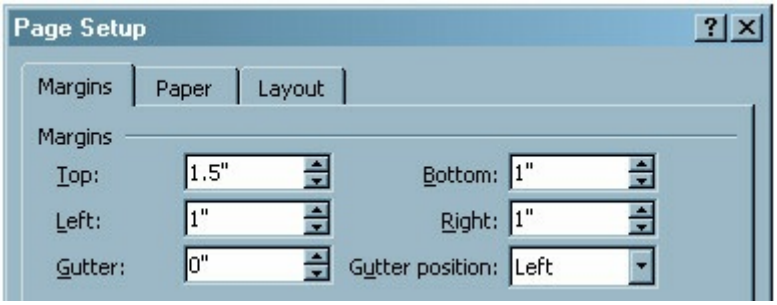

Page Setup
Margins
Paper
Layout
Margins
Top:
0.5
Bottom:
0.5
Left:
0.5
Right:
0.5
Gutter:
0"
Gutter position:
Left

Figure 4.19


Page Setup
Margins
Paper
Layout
Margins
Top:
1.5"
Bottom:
1"
Left:
1"
Right:
1"
Gutter:
0"
Gutter position:
Left

Creating Multiple Columns on a Page

SAS procedures produce consistent, readable output. However, they are not always good at using space efficiently. For example, the REPORT procedure displays a three-variable table the same way it displays a two-variable table, so long as it fits into the page width. However, when you look at the two-variable table, you realize that it wastes a lot of space. SAS takes a tall, skinny table and centers it in the middle of the page (or on the left if you're using the NOCENTER option). An example of this is shown in Figure 4.20. Here is the code that creates Figure 4.20:

```
      ODS RTF FILE='TwoColumn.rtf';
proc report data=clips nowd;
  column media mentions compete;
  define media / group;
  define mentions / display;
        define compete / display;
run;
ODS RTF CLOSE;
```

With ODS, you can take more control over your output. When you realize that you've got a tall, skinny table, one thing you can do is display it in two columns so that you get twice as much data on a page. The syntax is simple: just add the COLUMNS= option to the ODS RTF statement.

```
      ODS RTF FILE='TwoColumn.rtf' COLUMNS=2;
```

With the setting COLUMNS=2, the preceding PROC REPORT step produces the table shown in Figure 4.21. Now the page is being used more effectively to fit the output. Also, notice that the table headings repeat at the top of the new column.

However, there are some limitations to keep in mind. First of all, though you can set COLUMNS= to larger numbers, you will probably want to limit your output to two columns for portrait-oriented pages and three columns for landscape-oriented pages. Too many columns just make the page hard to read. Another limitation to keep in mind is that when you use the COLUMNS= option, it's up to you to determine whether your output will fit into the multiple columns. If your table has too many variables, SAS will fit it into two columns, but the results will not be pretty. Take the example of the following PROC PRINT step that lists the observations in the CLINICAL data set. This report lists nine variables and the observation row number for each observation.

```
      ODS RTF FILE='TwoColumnClinical.rtf' COLUMNS=2;
proc print data=clinical;
run;
ODS RTF CLOSE;
```

The COLUMNS= option works for other destinations as well. However, it doesn't make much sense for HTML output, because HTML pages can be as wide and as long as they need to be.

Unfortunately, when ODS writes this PROC PRINT report in a two-column format, the first

part of the report can fit only the first five variables in the column, while in the rest of the report, the remaining four variables fit in a column. SAS takes the first five variables and prints their values in two columns until it lists all of the observations in CLINICAL. Then it starts over with the remaining four variables. Unfortunately, it starts the set of four variables wherever it finished with the first five variables, even if that is in the middle of the page. Figure 4.22 shows the break between the two parts of the table, which occurs in the middle of a page.

Figure 4.20

Media	Number of references to company	Number of references to competition
Magazine	8	5
	11	5
	2	2
	7	5
	1	2

Figure 4.21

Media	Number of references to company	Number of references to competition
Magazine	8	5
	11	5
	2	2
	7	5
	1	2
	9	5

Media	Number of references to company	Number of references to competition
	3	1
	7	2
	6	1
	5	2
	10	6
	9	3

Figure 4.22

Obs	tx	age	gender	basedbp	basesbp
217	1	28	Male	114	136
218	1	37	Female	109	140
219	3	30	Female	80	171
220	2	28	Female	98	136

Obs	postdbp	postsbp	chgsbp	chgdbp
1	62	101	-12	-13
2	89	129	22	2
3	70	107	-18	-20
4	79	113	-58	-21
5	86	119	-40	-22

Obs	postdbp	postsbp	chgsbp	chgdbp
31	84	128	27	5
32	46	93	-18	-32
33	74	117	-22	-13
34	92	116	-35	-25
35	95	106	-63	4
36	68	114	-40	-44
37	92	105	-14	-20
38	89	119	17	2
39	84	117	-10	2
40	59	106	-57	-13
41	86	117	-14	-4

Adding “Page X of Y” to RTF Reports

When you are producing multi-page reports, it’s useful to be able to indicate how many pages the reader should expect. This avoids problems caused by someone not receiving all of the pages. If your SAS reports have page numbers in the format of “Page 1 of 3”, “Page 2 of 3”, etc., and a page gets lost in a fax transmission, the receiver will know to call you back and ask for the missing page.

Standard SAS page numbering just gives you a single page number. For example, Figure 4.23 shows the top of an RTF file created by ODS. The file contains a PROC PRINT report of the data set CLIPS. This is the first page of the output, so the page number 1 is generated in the top right header. This page number was generated by ensuring that page numbering was turned on with the following SAS system option setting:

```
OPTIONS NUMBER;
```

These page numbers are implemented in word processors by using a field that automatically computes the page number. This computation is useful because if you later edit your ODS RTF output, the page numbering will adjust automatically. In Word, you can use ALT-F9 to toggle the field so that it shows the code instead of the page number. This is shown in Figure 4.24.

The same word processor fields are used to generate page numbers in the “Page X of Y” format. To ask ODS to create those fields, you need to turn off page numbering with the NONUMBER SAS system option, and you need to add an escape sequence to your TITLE statement that requests the “Page X of Y” text. The following code adds the “Page X of Y” text to the PROC PRINT report. The code centers the title text “Data Set CLIPS”, and it right-justifies the “Page X of Y” text. If you use the BODYTITLE option, ODS will not place the “Page X of Y” text in the page header of the RTF document.

```
OPTIONS NONUMBER;
ODS ESCAPECHAR='^';
title justify=center 'Data Set CLIPS'
    justify=right 'Page ^{pageof}';
```

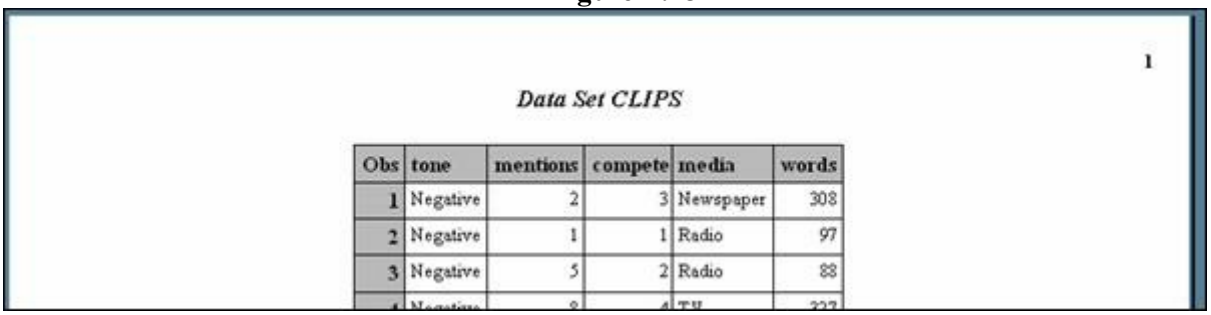
For the RTF destination, SAS gives you a simple escape sequence to use to request the “X of Y” page numbers. The {pageof} escape sequence requests that ODS insert the word processor fields to generate the current page number and the total number of pages. You can see the results in Figure 4.25. The field codes, which can be viewed by pressing ALT-F9 in Word, are shown in Figure 4.26.

This is a great way to add custom page numbers without having to know anything about RTF tags. However, this method has its limitations. The only page numbering format you can get with the {pageof} escape sequence is “X of Y”. If you’d like something different, like “X/Y”, you have to do that with different ODS escape character functions, as shown in the following specification. The {thispage} function returns the current page. The {lastpage} function returns the total number of pages in the document.

```
ODS ESCAPECHAR='^';
title justify=center 'Data Set CLIPS'
      justify=right 'Page ^{thispage}/^{lastpage}';
```

These ODS ESCAPECHAR functions generate the actual RTF control strings for the page number in the desired format. This is a better alternative than using the equivalent RTF control strings, because the RTF control strings are very fussy about space, slashes, curly braces, and line breaks. If you mistype the RTF control strings, you might create an RTF file that cannot be opened.

Figure 4.23

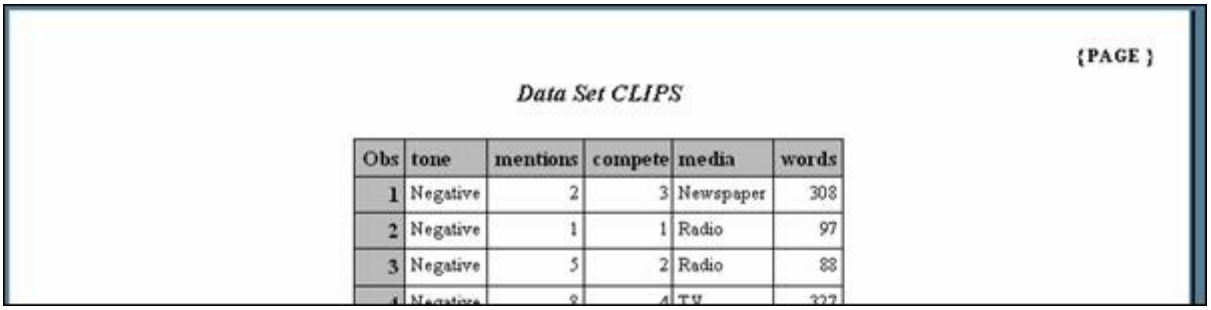


1

Data Set CLIPS

Obs	tone	mentions	compete	media	words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Negative	5	2	Radio	88
4	Negative	8	4	TV	327

Figure 4.24

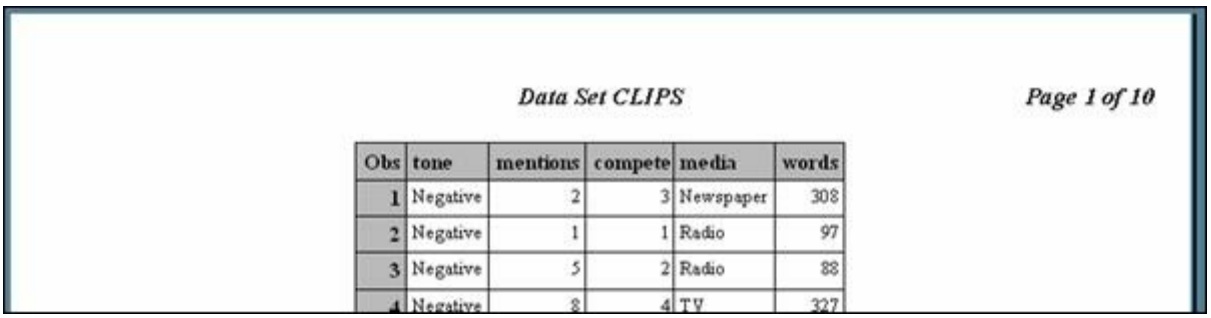


{PAGE }

Data Set CLIPS

Obs	tone	mentions	compete	media	words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Negative	5	2	Radio	88
4	Negative	8	4	TV	327

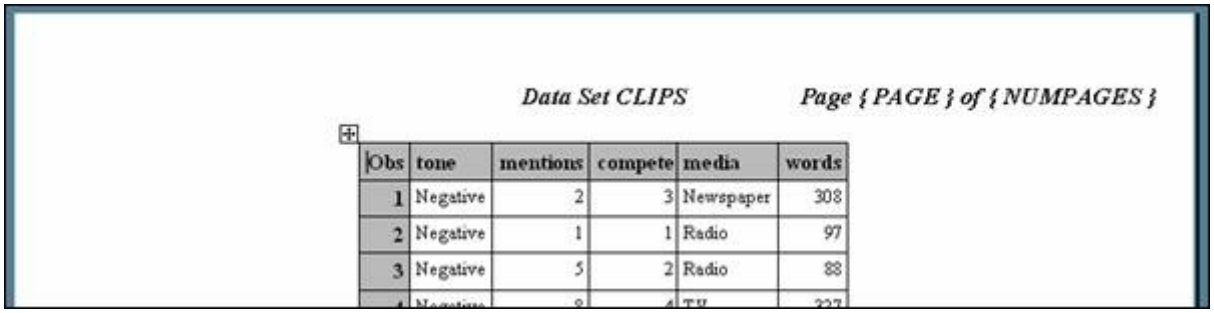
Figure 4.25



Data Set CLIPS *Page 1 of 10*

Obs	tone	mentions	compete	media	words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Negative	5	2	Radio	88
4	Negative	8	4	TV	327

Figure 4.26



Data Set CLIPS *Page { PAGE } of { NUMPAGES }*

Obs	tone	mentions	compete	media	words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Negative	5	2	Radio	88
4	Negative	8	4	TV	327

Adding Static or Dynamic Dates

Adding a date to a header or footer provides useful information about when a file was created or modified. By default, ODS inserts an RTF control string that corresponds to the Word date field in the page header of RTF output. This means that the date value is updated to the current date each time the file is opened.

Depending on why the date is needed, this might be what you want. However, for some reports, you want the date in the header to be the date the report was printed or delivered. For other reports, it might be preferable to have a fixed date that represents the date the report was generated.

With ODS RTF, you can create either static or dynamic dates. To generate a page header with a dynamic date that is updated every time the file is opened, you don't need to do anything. As long as the default SAS system option of DATE is in effect, you will get dynamic dates in your output. For example, Figure 4.27 shows the top of an RTF file created by ODS. The file contains a PROC PRINT report of the data set CLIPS. This view shows the field code that is generated. (In Word, press
ALT-F9 to toggle the field codes on and off.) Figure 4.28 shows the date generated by this field code. The SAS system option NUMBER is off, so page numbers are not shown in these two displays.

To generate a page header with a static date based on the day the output was generated, all you have to do is add the SASDATE option to the ODS RTF statement. When you add this option, SAS inserts text for the current date when the program is run. Remember that the SAS date is actually the time and date that you started your SAS session. So if you keep your session open for days at a time, this date might not correspond to the actual date the program was run. If you start a SAS session or batch process for your reports, then using the SASDATE option results in a static date that corresponds to the date the program was run.

```
ODS RTF FILE='StaticDate.rtf' SASDATE;
```

ODS writes this date as a fixed value. If you toggle to the field codes view by pressing ALT-F9, the view of the value remains the same, and no field code is displayed.

In order for either static or dynamic dates to be generated automatically the default system option of DATE must be in effect. However, neither of these date options gives you any control over the format of the date. The default dynamic or static dates are always in the format "hh:mm dddd, MMMM dd yyy".

Figure 4.27

{DATE \@ "hh:mm dddd, MMMM dd, yyyy " }

Data Set CLIPS

Obs	tone	mentions	compete	media	words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Neutral	6	1	Web	1018

Figure 4.28

11:08 Thursday, September 25, 2008

Data Set CLIPS

Obs	tone	mentions	compete	media	words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Neutral	6	1	Web	1018

Adding Customized Dates

What if you want a different format than “hh:mm dddd, MMMM dd yyyy”? And what if you want both a dynamic date that indicates when the file was last opened, and a static date that indicates when the file was created? You can do this, but you’ll need to turn off the system dates and create your own, as demonstrated in the following code:

```
      OPTIONS NODATE;
%let sdate=%sysfunc(putn("&sysdate"d,worddate.));
title justify=right '{Current date: \field {\*\fldinst { DATE \\@ "MMMM d, yyyy" \*\MERGEFORMAT}}}';
title2 justify=right "Report generated on: &sdate";
ODS RTF FILE='BothDates.rtf';
proc print data=clips;
      run;
ODS RTF CLOSE;
```

The %LET statement creates the macro variable SDATE, whose value is inserted as the static date value in the TITLE2 statement. The SAS language function PUTN formats the current date with the WORDDATE format. The %SYSFUNC macro language function executes the PUTN function and stores the result in the macro variable SDATE.

The TITLE statement contains the text and RTF control strings or Word field tags for the dynamic date. Refer to the sections “Enhancing RTF Output by Using RTF Control Words and Field Codes” and “Sending RTF Field Codes and Control Strings Only to RTF Output” later in this chapter for more information about specifying RTF field tags or control strings. These RTF control strings are very fussy about spaces, slashes, curly braces, and line breaks. A line break was added so that the code would fit on this page, but actually there should be only one space between the “@” that ends the first line and the double quotation mark that starts the second line.

Both date values are in the same format: “MMMM d, yyyy”. Both titles are right-justified with the JUSTIFY=RIGHT option on the TITLE statements.

The resulting output is shown in Figure 4.29. Both title lines show the same date (until the next time the file is opened, when the dynamic date in the first title is updated). Figure 4.30 shows the field codes.

Figure 4.29

Current date: September 25, 2008
Report generated on: September 25, 2008

Obs	tone	mentions	compete	media	words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Neutral	6	1	Web	1018

Figure 4.30

Current date: { DATE \@ "MMMM d, yyyy" }
Report generated on: September 25, 2008

Obs	tone	mentions	compete	media	words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Neutral	6	1	Web	1018
4	Positive	5	3	Magazine	561

Placing Title and Footnote Text in the Body of the Document

When ODS creates an RTF document, it places title text and footnote text by default in the header and footer sections of the document. If you want to place this text within the body of your document instead, you can add the BODYTITLE option or BODYTITLE_AUX option to your ODS RTF statement.

The difference between the two options is that the BODYTITLE option places title text and footnote text each in its own paragraph within the body of the document, whereas the BODYTITLE_AUX option places title text and footnote text each in its own table within the body of the document.

The following program demonstrates the default placement of title text and footnote text. Figure 4.31 presents the results in Print Preview mode.

```
ODS RTF FILE='DefaultPlacement.rtf';
title "Default Title Placement without BODYTITLE";
footnote "Default Footnote Placement without BODYTITLE";
proc tabulate data=clips format=8.;
  class media;
  var words;
  table min p50 max,words*media;
run;
ODS RTF CLOSE;
```

The BODYTITLE option on the following ODS statement causes ODS to place the title text and the footnote text each in its own paragraph within the body of the document. They appear before and after the PROC TABULATE report. The TITLE and FOOTNOTE statements are included. The PROC TABULATE step is identical to the previous PROC TABULATE step. Figure 4.32 presents the results in Print Preview mode.

```
ODS RTF FILE='Bodytitle.rtf' BODYTITLE;
title "Title Placement with BODYTITLE Option";
footnote "Footnote Placement with BODYTITLE Option";
```

The BODYTITLE_AUX option on the following ODS statement causes ODS to place the title text and the footnote text each in its own table within the body of the document. They appear before and after the PROC TABULATE report. The TITLE and FOOTNOTE statements are included. The PROC TABULATE step is identical to the previous PROC TABULATE step. Figure 4.33 presents the results with table gridlines shown. The gridlines show that the title text and footnote text are each in a one row, one column table.

```
ODS RTF FILE='Bodytitle_Aux.rtf' BODYTITLE_AUX;
title "Title Placement with BODYTITLE_AUX Option";
footnote "Footnote Placement with BODYTITLE_AUX Option";
```

Figure 4.31

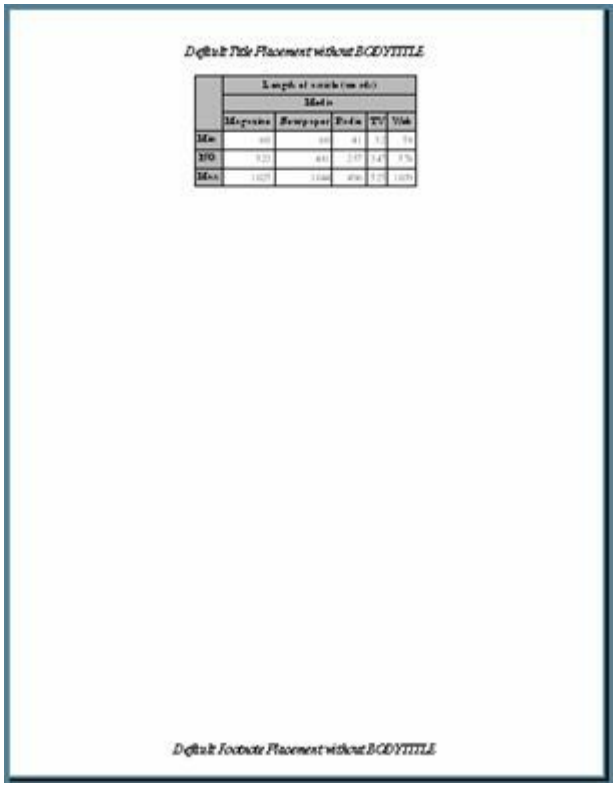


Figure 4.32

Figure 4.33

Title Placement with BODYTITLE_AUX Option

	Length of article (words)				
	Media				
	Magazine	Newspaper	Radio	TV	Web
Min	60	60	41	32	58
P50	522	481	257	347	576
Max	1025	1044	496	525	1039

Footnote Placement with BODYTITLE_AUX Option

Adding Images to RTF Output

One of the quickest and easiest ways to dress up your RTF reports is to drop in a logo or graphic on each page. All you need to do is find a JPEG or PNG or other type of graphic and add a few special instructions to use your graphic. We've already seen the use of ODS ESCAPECHAR to control the page numbering. Now we're going to use it to insert a logo in the SAS title. We're using the older syntax that works in either SAS 9.1.3 or SAS 9.2, because the newer function-call version of the ESCAPECHAR syntax does not allow the PREIMAGE= attribute inside the style function call.

The PREIMAGE= attribute in the following TITLE statement specifies the filename of an image to use in the title. The filename specification must include path information that points to the location of the graphic file when the program is run. Because ODS embeds the image in the RTF file, you do not need to worry about keeping the graphics file along with the output file, as you would if you had sent the output to the HTML destination.

```
      ODS NOPROCTITLE;
ODS RTF FILE='ToneFreqs.rtf';
ODS ESCAPECHAR='^';
title '^S={preimage="logo.jpg"}';
proc freq data=clips;
  tables tone;
run;
ODS RTF CLOSE;
```

Remember that the SAS title, by default, is put into the Header area of the document when you open the file with Microsoft Word. The Header area is shown in Figure 4.34. ODS centers the graphic in the Header area. If you do not want the logo in the document header and instead want it placed in the document body, you can run the same code and add the BODYTITLE option to the ODS RTF invocation statement as shown below. The logo would then appear in the body of the document.

```
      ODS NOPROCTITLE;
ODS RTF FILE='ToneFreqs.rtf' BODYTITLE;
<same code and TTITLE statement as above>
ODS RTF CLOSE;
```

Assume that the SAS system option CENTER is in effect. This setting centers both the output and the graphic as shown in Figure 4.35. You can allow SAS to place your graphic based on the SAS system option setting, or you can specify a justification setting in your style. The following code adds the JUST= option and additional tables to the PROC FREQ step. The justification attribute JUST= specifies a value of L for left. Two pages of output are shown in Figure 4.36, which not only shows the left-justified logo, but also shows that the logo appears at the top of every page.

```
title '^S={just=l preimage="logo.jpg"}';
```

If you want the logo specification to be separate from the procedure code, you can change the style template to use the PREIMAGE= attribute. Examples of this technique are shown in Chapter 18, "Doing More with Styles and ODS."

Figure 4.34: Logo in Header

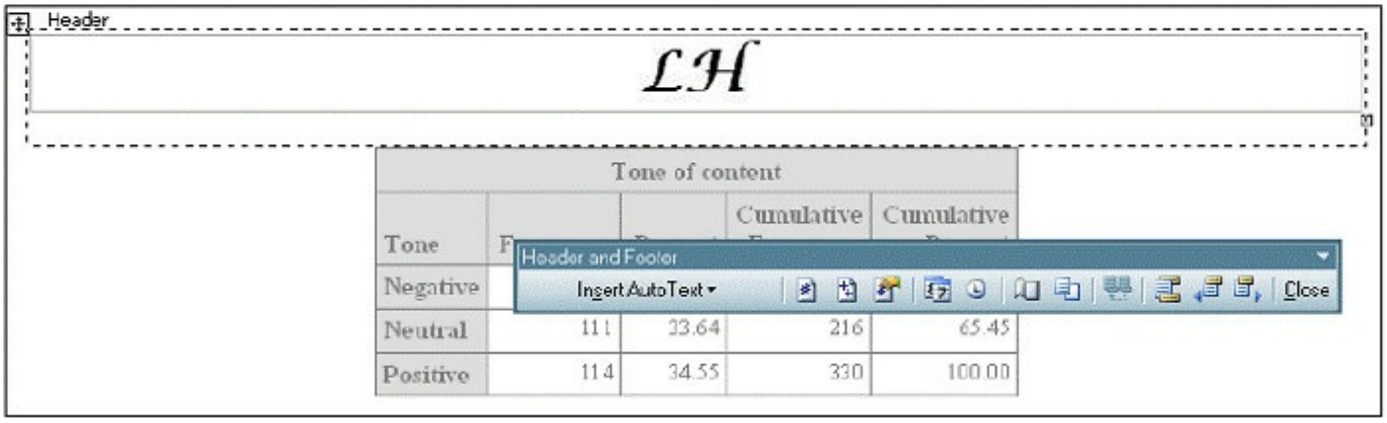


Figure 4.35: Logo in Body

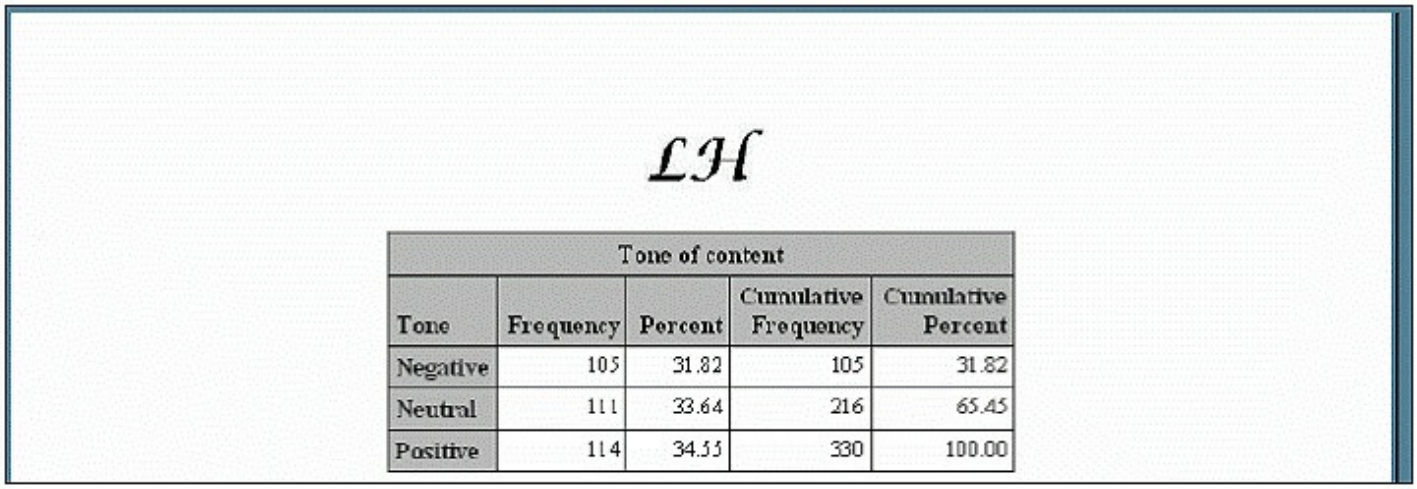
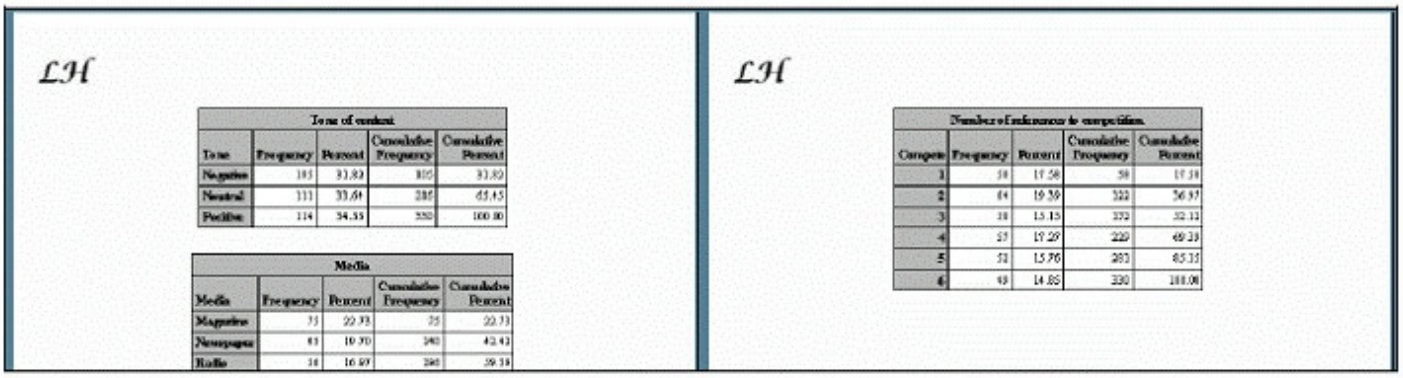


Figure 4.36: Logo Left-Justified on Two Pages in Print Preview Mode



Adding Text

One of the reasons RTF is such a useful output destination is that we can cut and paste from the RTF document or we can add explanatory text to our RTF document to build our final report. But wouldn't it be nice if the entire report could be generated automatically?

To insert text into an RTF document, use the TEXT= option on an ODS statement to write to all open destinations, including RTF. The following statement inserts text into an open RTF destination file:

```
ODS TEXT='Text to insert';
```

You also need to add the STARTPAGE=OFF option to your ODS statement to prevent SAS from inserting page breaks between your table and text.

The output from the following program contains results from one ODS table from the PROC CORR step and adds text below the table. The ODS TEXT= statement follows the PROC CORR step. The text wraps across several lines, and only one statement is needed.

An escape character is defined at the beginning of the program. It is used to signal to ODS to apply left and right margins of one inch to the text. Figure 4.37 presents the output from this program.

```
    ODS ESCAPECHAR='^';
ODS SELECT PearsonCorr;
ODS RTF FILE='AddText.rtf' STARTPAGE=OFF;
proc corr data=clips nosimple;
  var mentions;
  with compete;
run;
ODS TEXT='^S={LEFTMARGIN=1in RIGHTMARGIN=1in}
This text was inserted with the TEXT= option. It can be used to add a detailed explanatory caption below a table or
figure. This example uses an inline formatting escape sequence to apply different styles here. This text can also be
formatted by using PROC TEMPLATE and the USERTEXT style element';
ODS RTF CLOSE;
```

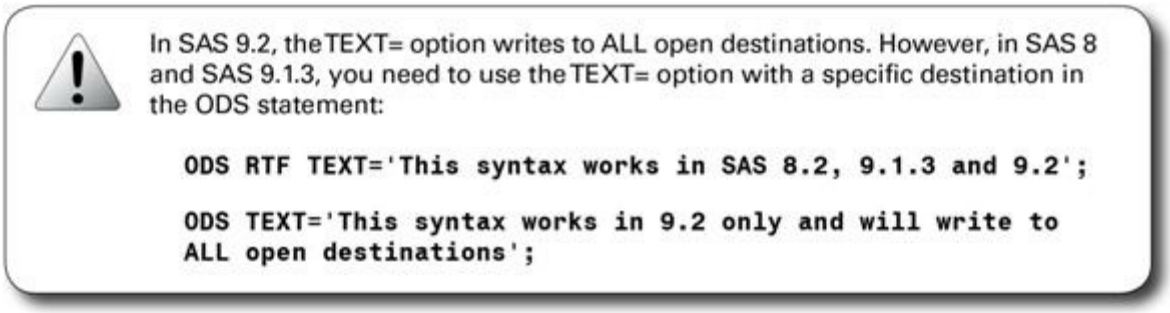


Figure 4.37

The CORR Procedure

Pearson Correlation Coefficients, N = 330 Prob > |r| under H0: Rho=0	
	Mentions
Compete Number of references to competition	0.14996 0.0063

This text was inserted with the TEXT= option. It can be used to add a detailed explanatory caption below a table or figure. This example uses an inline formatting escape sequence to apply different styles here. This text can also be formatted by using PROC TEMPLATE and the USERTEXT style element

Adding More Complex Text

To generate a more complicated report, just add more procedures and more TEXT= options. The following code generates Figure 4.38. Since we showed ODS TEXT= in the previous example, we show the other version of the TEXT= option in this example. This version of the TEXT= option writes *only* to the RTF destination, as opposed to the previous statement, which writes to ALL open destinations.

```
ODS ESCAPECHAR='^';

ODS SELECT PearsonCorr;
ODS RTF FILE='c:\temp\ReportWithText_4_38.rtf' STARTPAGE=OFF;
ODS RTF TEXT='^{style[font_style=italic just=c fontweight=bold fontsize=14pt]This bold, italicized text in 14pt introduces the table. This introductory text will automatically wrap.}';
proc corr data=clips nosimple;
   var mentions;
   with compete;
run;
ODS RTF TEXT='^{newline 4}';
ODS RTF TEXT='^{style[font=(Arial)]Four lines appear before this sentence (in Arial) and a bar chart follows:}';
proc gchart data=clips;
  vbar mentions / midpoints=1 to 12 by 1 inside=freq;
run;
quit;
ODS RTF TEXT='^{style[fontweight=bold textdecoration=underline]This bold, underlined sentence concludes the report.}';
ODS RTF CLOSE;
```

If you wanted to customize the text, you could, for example, use macro variables to add the current date or insert custom text based on the results in the data. One thing to keep in mind is that as the text gets lengthy, you will start to get warning messages in the log about the string being longer than 512 characters. These messages can be ignored or turned off with the NOQUOTELENMAX option.

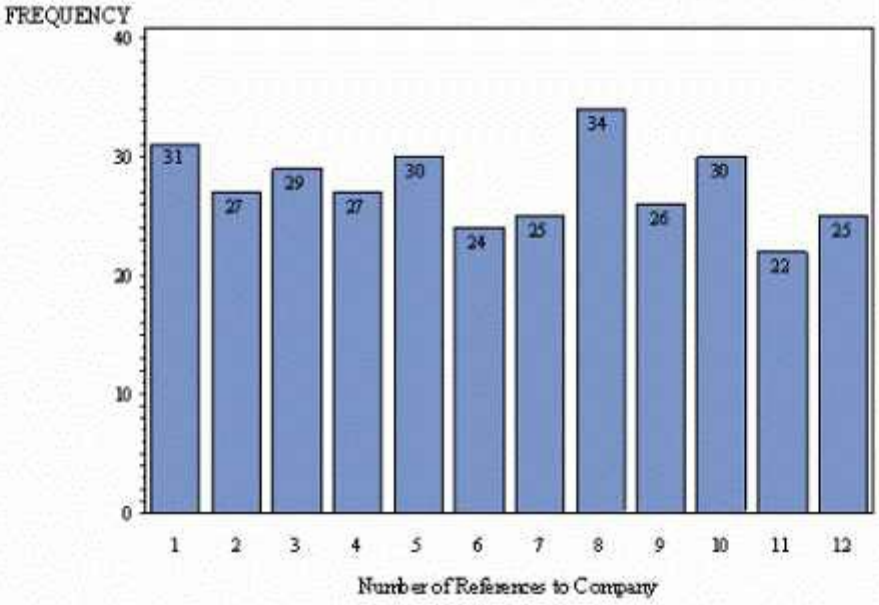
Figure 4.38

This bold, italicized text in 14pt introduces the table. This introductory text will automatically wrap.

The CORR Procedure

Pearson Correlation Coefficients, N = 330 Prob > |r| under H0: Rho=0	
	mentions
compete Number of References to Competition	0.14996 0.0063

Four lines appear before this sentence (in Arial) and a bar chart follows:



This bold, underlined sentence concludes the report.

Adding Hyperlinks to RTF Output

Web pages aren't the only place you can use hyperlinks. You can put a link in a word processing document so that the reader can link to related information. With word processing documents frequently being shared electronically rather than on paper, this feature can be useful.

Adding a hyperlink is simple. You can add this link anywhere you have text in your output, such as labels, formatted values, titles, and footnotes. All you need to do is add the following escape sequence, where the escape character has been defined as the caret (^):

```
^S={URL="http://www.myweb.com"}
```

This code applies the style attribute URL= to the subsequent text, which specifies a destination for the hyperlink. If you want to use the new form of ODS ESCAPECHAR syntax to set the URL for RTF files, you have to use the TAGSETS.RTF destination, as shown later in this chapter. We show here the original form of the style override syntax to specify the text on our ODS RTF TEXT= statement because you might have legacy code that uses this technique. Keep in mind that this original ODS ESCAPECHAR method applies the URL value to the whole cell and not just to a single word or sentence inside the cell, and it only works if the escape character (in this case, '^') is the first character in the text string.

The following example adds a hyperlink at the bottom of the output using the ODS RTF TEXT= statement. Rather than putting the hyperlink in a footnote, which would be at the bottom of the page, the TEXT= feature places the hyperlink in a text box right below the procedure output. Remember that if we used the ODS TEXT= form of the option, the text would be written to all open destinations.

```
    ODS ESCAPECHAR='^';
ODS RTF FILE='Hyperlink.rtf';
proc freq data=clips;
     table compete;
    run;
ODS RTF TEXT='^S={URL="http://www.myweb.com"}Source: www.myweb.com';
ODS RTF CLOSE;
```

The resulting RTF file is shown in Figure 4.39. You can see the text for the hyperlink below the table. If you move the mouse over the text, as in Figure 4.40, you can see that it is a hyperlink. However, it's not obvious to readers that a hyperlink is there, unless they happen to move the mouse over the correct part of the page.

To solve this problem, you can provide a visual clue for the reader by changing the text color of the hyperlink to blue, the typical color used for hyperlinks. To do that, add a second escape sequence that sets the color of the text for the Web site address to blue, as in the following code:

```
ODS RTF TEXT='^S={URL="http://www.myweb.com"}Source: ^S={FOREGROUND=blue}www.myweb.com';
```

The revised output is shown in Figure 4.41. Now the Web site address looks like a

standard hyperlink because it appears in blue (like a Web page hyperlink), which should prompt the user to click on the link.

Figure 4.39

Number of references to competition				
Compete	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	58	17.58	58	17.58
2	64	19.39	122	36.97
3	50	15.15	172	52.12
4	57	17.27	229	69.39
5	52	15.76	281	85.15
6	49	14.85	330	100.00

Source: www.myweb.com

Figure 4.40

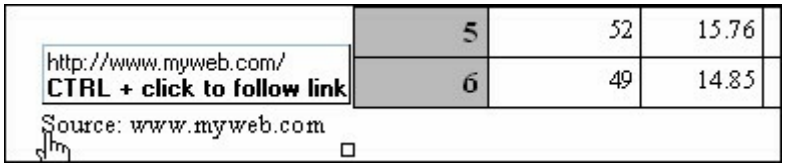


Figure 4.41

Number of references to competition				
Compete	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	58	17.58	58	17.58
2	64	19.39	122	36.97
3	50	15.15	172	52.12
4	57	17.27	229	69.39
5	52	15.76	281	85.15
6	49	14.85	330	100.00

Source: www.myweb.com

Adding a Table of Contents to RTF Output

Starting with SAS®9, you can direct ODS to specify a table of contents in the RTF document that it creates. It does this by inserting control fields in your RTF document that your word processing software can use to build a table of contents. In SAS 8, you had to insert the RTF control fields yourself or build the table of contents manually from inside the word processing software.

In SAS 9.2, you add the CONTENTS= and TOC_DATA options to the ODS RTF statement to request a table of contents in the output.

```
ODS RTF FILE='PayrollTOC.rtf' CONTENTS=yes TOC_DATA;
```

This causes SAS to insert a table of contents as the first page of the report, as shown in Figure 4.42, and to insert control fields that support the table of contents in the pages of the report. Figure 4.43 shows these control fields in a typical report page.

The following program created Figures 4.42 and 4.43. The ODS PROCLABEL statement, which is described in Chapter 3, “HTML Output,” replaces the procedure title with the text specified on the statement. Note that the labels for the page dimension in the PROC TABULATE step become references in the table of contents.

```
ODS RTF FILE='PayrollTOC.rtf' CONTENTS=yes TOC_DATA;
ODS PROCLABEL 'Human Resources Report on Salary and Hours';
proc tabulate data=hr;
  class department category;
  var annualsalary hoursweek;
  table department='Annual Salary Statistics for '
   all='Annual Salary Statistics in All Departments and Categories',
        (category all='All Categories in Department'),
        annualsalary*( (n mean min median max)*f=dollar8.);
  table department='Hours per Week Statistics for ' all=
     'Hours per Week Statistics for All Departments and Categories',
        (category all='All Categories in Department'),
        hoursweek*( (n mean min median max)*f=3.);
run;
ODS RTF CLOSE;
```

When you run this code and open the resulting file, the table of contents does not automatically appear. Turn the page to learn how to use the word processor to complete the final step of generating the table of contents.

Figure 4.42

Table of Contents

Human Resources Report on Salary and Hours 2
Cross-tabular summary report 2
Table 1 2
Annual Salary Statistics for Administration 2
Annual Salary Statistics for Manufacturing 3
Annual Salary Statistics for Marketing 4
Annual Salary Statistics for R&D 5
Annual Salary Statistics in All Departments and Categories 6
Table 2 7
Hours per Week Statistics for Administration 7
Hours per Week Statistics for Manufacturing 8
Hours per Week Statistics for Marketing 9
Hours per Week Statistics for R&D 10
Hours per Week Statistics for All Departments and Categories 11

Section Break (Next Page)

(This screen shot has the Word Show/Hide Formatting option toggled ON.)

Figure 4.43

{tc "Human Resources Report on Salary and Hours" \f C \l 1}{tc "Cross-tabular summary report" \f C \l 2}{tc "Table 1" \f C \l 3}{tc "Annual Salary Statistics for Administration" \f C \l 3}

Annual Salary Statistics for Administration

	Annual Salary				
	N	Mean	Min	Median	Max
Position Category					
Contract	$14	$47,500	$35,000	$35,000	$99,500
Hourly	$20	$50,600	$35,000	$36,000	$126,500
Salaried	$36	$52,611	$35,000	$39,250	$143,500
All Categories in Department	$70	$51,014	$35,000	$35,500	$143,500

Section Break (Next Page)

(This screen shot shows the Word hidden formatting fields that produce the TOC.)

Completing the Table of Contents

When you first open the RTF output file in Microsoft Word, it looks as though ODS has failed you because it displays a front page with the heading "Table of Contents" but no actual table of contents. The reason for this is that ODS does not determine pagination, but instead hands this task off to the word processor. ODS does not build a table of contents because it does not know the page numbers to insert into the table.

You must direct your word processing software to complete the table of contents. To do this after opening the RTF file, click the Show/Hide paragraph marker (¶) to turn on formatting markers if they are not already displayed. Next, position your cursor at the edge of the section break following the table of contents heading, right-click, and select **Update Field**, which is shown in Figure 4.44. This causes Microsoft Word to build and insert the table of contents, which is shown in Figure 4.42 on the previous page.

If you add text or page breaks after the table of contents is initially built, you can follow these same steps to update the page numbers or to rebuild the table.

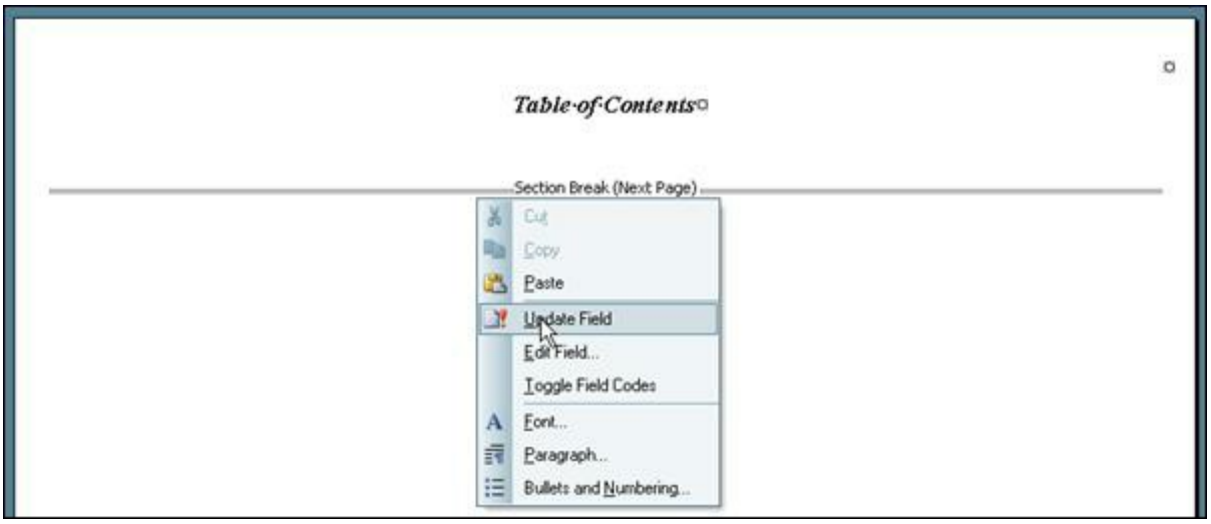
In SAS 9.1.3, you add the CONTENTS= option to the ODS RTF statement to request a table of contents in the output. The TOC_DATA option is the default setting for SAS 9.1.3, so you do not have to include it in your code. Here is the syntax to generate a separate contents page in SAS 9.1.3:

```
ODS RTF FILE='PayrollTOC.rtf' CONTENTS=yes;
```

When you know you do not want a table of contents, you can suppress the insertion of the table of contents control words by adding the NOTOC_DATA option to your ODS RTF statement.

```
ODS RTF FILE='NoTOC.rtf' NOTOC_DATA;
```

Figure 4.44



Specifying Metadata in Your RTF File

The ODS HTML statement options HEADTEXT= and METATEXT= can insert information into the <HEAD> section of your HTML file. This kind of information about your output file is called metadata. The ODS RTF statement also has options that control the insertion of metadata information into your RTF document. It is then up to your word processor to use the metadata information.

The kinds of metadata that you can tell ODS to store in your RTF document includes author name, document title, and operator name. This information is visible in Microsoft Word when you view the properties of the RTF document. To view the properties, open your document and select **Properties** from the **File** menu in Word 2003, or select **Prepare** u **Properties** from the Office Button menu in Word 2007.

The following SAS 9.2 program specifies the three metadata options available on the ODS RTF statement: AUTHOR=, TITLE=, and OPERATOR=. The PROC MEANS step is identical to the one that produces Figure 4.1, and the report is not shown with this example.

```
ODS RTF FILE='Payroll_Meta.rtf'
        AUTHOR="HR Report Designer"
        TITLE="HR Standard Payroll Report"
        OPERATOR="HR Assistant";
title 'Payroll Report';
proc means data=hr nonobs maxdec=0 mean sum;
  class Department;
  var AnnualSalary;
run;
ODS RTF CLOSE;
```

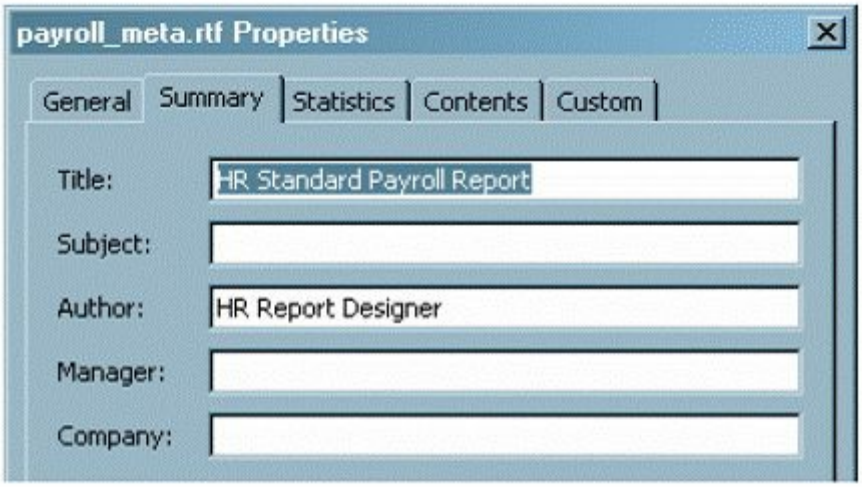
These are the generated RTF control strings, which are put into your document:

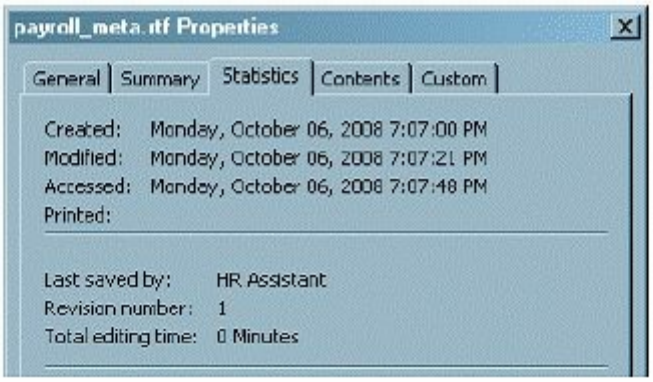
```
{\info{\title HR Standard Payroll Report}{\author HR Report Designer}
{\operator HR Assistant}
```

It is then up to the word processor (Word) to render the control strings appropriately. Figure 4.45 shows the properties of the file created by the preceding program. The **Summary** tab shows the title and author information. The **Statistics** tab shows that the value specified by the OPERATOR= option is inserted in the **Last saved by** field.

The author, title, and operator control strings are not visible in the document itself. Therefore, you don't have to worry about turning them on or off as you did with contents control strings in the previous example.

Figure 4.45: Metadata Information Shown in Microsoft Word





Aligning Decimals in RTF Output

Before SAS®9, the only way to align data values in a column by decimal points when creating RTF output with ODS was to insert the RTF control word \tqdec in the PRETEXT= attribute of the STYLE= option in PROC PRINT, PROC REPORT, or PROC TABULATE.

Starting with SAS®9, the JUST= style attribute can have the value D for decimal justification. The following PROC REPORT step applies this new setting to the cells in its report on the data set CHEMTEST. The variable SAMPLE is a character variable with a period in the values. The variable CHEMPPB is a numeric variable with widely ranging values.

Figure 4.46 shows how the output would look if the two STYLE= options were omitted from the two DEFINE statements in the PROC REPORT step. Figure 4.47 shows the output with the two STYLE= options included. RTF control words are not needed to align the values as shown in Figure 4.47.

```
ODS RTF FILE='DecimalAlign.rtf';
proc report data=chemtest nowd;
  title 'Decimals Aligned';
  column sample chemppb;
  define sample / display style(COLUMN)={just=d};
  define chemppb / display style(COLUMN)={just=d};
run;
ODS RTF CLOSE;
```

The decimal justification attribute is supported in the RTF and PDF destinations. However, if you sent the results from the preceding code to the HTML destination, you would see that the JUST=D attribute is not used in the HTML file, as shown in Figure 4.48.

Figure 4.46: Default Alignment

Decimals Not Aligned

Sample	ChemPPB
lab.57175	1250.3
desk.71309	2530
lab.28009	0.94
warehouse.40035	14.30185
truck.55128	390
desk.41930	.
unknown	0.00087

Figure 4.47: RTF Report Using JUST=D Attribute

Decimals Aligned

Sample	ChemPPB
lab.57175	1250.3
desk.71309	2530
lab.28009	0.94
warehouse.40035	14.30185
truck.55128	390
desk.41930	.
unknown	0.00087

Figure 4.48: HTML Report Using JUST=D Attribute

Decimals Aligned

Sample	ChemPPB
lab.57175	1250.3
desk.71309	2530
lab.28009	0.94
warehouse.40035	14.30185
truck.55128	390
desk.41930	.
unknown	0.00087

Exploring RTF as Created by Microsoft Word

One way to understand RTF control words is to read the Microsoft documentation on the subject. Another approach is to create a file with Microsoft Word using the formatting that you want and save the file as an RTF file. You can then open the RTF file with Notepad or another text editor to view and edit the control words. This is especially helpful if you want to do something special like insert some other formatting or information.

In a previous example, we populated three fields in the Word document properties by specifying values for the three ODS RTF options: AUTHOR=, TITLE=, and OPERATOR=. These are the only document properties options available with ODS RTF. Yet if we look at the document properties of an RTF file that we create from scratch, we see that there is an area for entering Keywords, Comments, Manager, and Subject. Refer to Figure 4.49 to see the properties of the SAMPLE.RTF file as they were set manually inside Microsoft Word. Compare Figure 4.49 to Figure 4.50 to see how those properties were saved using RTF control strings in the {\info} section. The highlighted strings in the first part of Figure 4.50 are the ones that we would like to be able to modify.

The only way to set document properties with ODS RTF is through the use of the three options AUTHOR=, TITLE=, and OPERATOR=. The technique shown in the following program inserts information into fields other than these three by "piggybacking" control words to the specifications for the three ODS RTF options. The program starts by defining six macro variables for six document properties fields. References to the macro variables are then added to two of the ODS RTF options. The macro variables are used solely to simplify the code. You can also directly key the control words into the specification for the ODS RTF option.

```
%let d = %str({\doccomm This report produced without errors});
%let k = %str({\keywords HR, Payroll, Department, Annual Salary});
%let s = %str({\subject Salary Statistics});
%let m = %str({\*\manager HR Manager});
%let c = %str({\*\category Internal});
%let n = %str({\*\company Global Company, Inc.});
ODS RTF FILE='Payroll_Meta2.rtf'
        AUTHOR="HR Report Designer &d &k &s"
        TITLE="HR Standard Payroll Report &m &c &n"
        OPERATOR="HR Assistant";
title 'Payroll Report';
proc means data=hr nonobs maxdec=0 mean sum;
  class Department;
  var AnnualSalary;
run;
ODS RTF CLOSE;
```

The document properties of the new results file are shown in Figure 4.49 for the Payroll_Meta2.rtf file. Its {\info} section is presented in the second part of Figure 4.50. It was necessary to refer to RTF documentation to determine the details of coding these control words. For example, curly braces around each property specification are required, and the extra '*' is required ahead of a "userprop" string. Also, you do not need to turn the

PROTECTSPECIALCHARS attribute to OFF in order to pass these strings into the RTF file.

Figure 4.49

Properties Set Manually in RTF	Properties Set with ODS RTF Code
Sample File.rtf Properties General | Summary | Statistics | Contents | Custom Title: Sample File Subject: Subject Field Author: Author Field Manager: Manager Field Company: Company Name Category: Category Field Keywords: Keyword1, Keyword2, Keyword3 Comments: Comments Comments Comments Comments Hyperlink base: Template: Normal.dot ☐ Save preview picture OK Cancel	Payroll_Meta2.rtf Properties General | Summary | Statistics | Contents | Custom Title: HR Standard Payroll Report Subject: Salary Statistics Author: HR Report Designer Manager: HR Manager Company: Global Company, Inc. Category: Internal Keywords: HR, Payroll, Department, Annual Salary Comments: This report produced without errors Hyperlink base: Template: Normal.dot ☐ Save preview picture OK Cancel

Figure 4.50: Investigating {\info} Section

Properties Set Manually

```
{\info{\title Sample File}{\subject Subject Field}{\author Author Field}{\keywords Keyword1, Keyword2, Keyword3}{\doccomm Comments Comments Comments Comments}{\operator abc}{\creatim\yr2008\mo10\dy7\hr16\min58}{\revtim\yr2008\mo10\dy7\hr16\min58}{\version3}{\edmins0}{\nofpages1}{\nofwords0}{\nofchars0}{\*\manager Manager Field}{\*\company Company Name}{\*\category Category Field}{\nofcharsws0}}
```

Properties Set with ODS RTF Code

```
{\info{\title HR Standard Payroll Report {\*\manager HR Manager} {\*\category Internal} {\*\company Global Company, Inc.}}{\author HR Report Designer {\doccomm This report produced without errors} {\keywords HR, Payroll, Department, Annual Salary} {\subject Salary Statistics}}{\operator HR Assistant}{\version1}{\creatim\yr2008\mo10\dy7\hr17\min22\sec15}}
```

Inserting Field Codes into Your RTF Output

Microsoft Word has *field codes*, such as the PAGE or NUMPAGES or DATE fields that we saw in previous examples. This example demonstrates how to use field codes to insert the filename or path of the RTF file into the footnote of an RTF document that ODS is creating. There are no ODS RTF statement options to accomplish this common task, so it is necessary to use field codes.

As seen in previous examples, it is necessary to turn off the protection of special characters such as the backward slash (\) and curly braces ({}) used by RTF. This is done either by setting an escape character and applying the style attribute PROTECTSPECIALCHARS=OFF before the first usage of the RTF control word in the statement (the SAS 9.1.3 method), or by setting the PROTECTSPECIALCHARS style attribute in a style template (the SAS 9.2 method).

Whether you use the ODS ESCAPECHAR method or the style template method, your output is the same. Since the SAS 9.2 template code is so simple, we'll show it here instead of deferring it to the template chapter. The following program executes a PROC MEANS step and shows why you need to turn off protection of the slash and curly braces. As seen in Figure 4.51, the RTF field codes do not produce the desired filename in the footnote when PROTECTSPECIALCHARS=ON (the default). The first FOOTNOTE statement starts with the RTF control word for left justification (\ql). The RTF control word that follows inserts the field code for FileName. However, you can see in Figure 4.51 that Microsoft Word has not interpreted the control words and field codes as anything but text. All the control words and field codes are shown in the footnote because ODS "protected" them.

```
ODS RTF file='Footnotes3.rtf';
ODS ESCAPECHAR='^';
footnote "\ql The file is: {\field{\*\fldinst { FileName }}}";
<PROC MEANS step>
ODS RTF CLOSE;
```

However, if you use this simple style template in SAS 9.2, with two subsequent PROC MEANS steps, the field codes produce the desired output.

```
proc template
  define style styles.usefc;
  parent=styles.rtf;
    class systemfooter/
       protectspecialchars=off;
  end;
run;
ODS RTF file='Footnotes_Temp.rtf' style=styles.usefc;
footnote "\ql The file is: {\field{\*\fldinst { FileName }}}";
<PROC MEANS step>
footnote "\ql The file is: {\field{\*\fldinst { FileName \\p}}}";
<PROC MEANS step>
ODS RTF CLOSE;
```

The only difference between the second and third FOOTNOTE statements is that the third

FOOTNOTE statement includes the switch to insert the path (\\p). Output from the second FOOTNOTE statement is shown in Figure 4.52 (in Word Print Preview mode). And output from the third FOOTNOTE statement is shown in Figure 4.53. The third footnote includes the path (assume the path was set to c:\temp).

One note about the field instructions: When you initially open the Word document, you will not see values for the field instructions in effect in the document until you either select **Print Preview** mode from the **File** menu in Microsoft Word 2003 or **Update the Field** when using Microsoft Word 2007. If you want more information about these and other field codes, you can consult the Microsoft Word documentation or just create a Word document and experiment. However, keep in mind that fields that are specific to Microsoft might not be honored by other word processor software.

An alternate technique, most often used in SAS 9.1.3, uses the field codes in a second and third FOOTNOTE statement by specifying an ODS escape character and the style attribute directly in the footnote statement.

```
ODS RTF file='Footnotes.rtf';
ODS ESCAPECHAR='^';
footnote "^S={protectspecialchars=off}\ql The file is: {\field{\*\fldinst { FileName }}}";
<PROC MEANS step>
footnote "^S={protectspecialchars=off}\ql The file is: {\field{\*\fldinst { FileName \\p}}}";
<PROC MEANS step>
ODS RTF CLOSE;
```

The second and third FOOTNOTE statements turn off protection of the special characters so that Microsoft Word interprets the control words and field codes. This is accomplished by setting PROTECTSPECIALCHARS=OFF at the beginning of each statement. Each statement starts with the escape character defined by the ODS ESCAPECHAR= statement followed by the style attribute specification of PROTECTSPECIALCHARS=OFF, and each produces the same output as the style template method. Although this alternate technique also works in SAS 9.2, the style template method gives you more flexibility to use the RTF field codes directly in your TITLE and FOOTNOTE statements without using ODS ESCAPECHAR=.

Figure 4.51: (Print Preview mode)

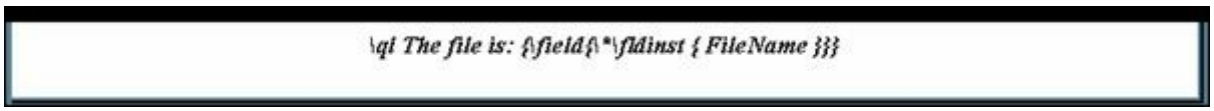


Figure 4.52: (Print Preview mode)

Figure 4.53: (Print Preview mode)

Enhancing RTF Output by Using RTF Control Words and Field Codes

Another method, which is described below, is to place RTF control words and field codes directly in the TITLE or FOOTNOTE statements in the program that creates your RTF document. There are so many control words that we show only a selection here. At the end of the chapter, see "More Enhancement of RTF Output Using RTF Control Words" for another example of using RTF control words.

Table 4.1 lists selected RTF control words that you might find useful. Figures 4.54 and 4.55 show the result of using these control words in a series of ten TITLE and three FOOTNOTE statements. Because the code is so lengthy, refer to the end of the chapter for the complete program code.

Table 4.1: Selected RTF Control Words

Style	RTF Control Word(s)	Quoted String in TITLE or FOOTNOTE Statement
Bold Italicize	\b \i	"\b\i0 Bold \b0 \i Italicize"
Bullet	\bullet	"\bullet Bullet"
Double strike	\striked1	"\striked1 Double Strike \striked0 Regular Words"
Double underline	\dul	"\uldb Double Underline \uldb0"
Engrave	\impr	"\impr Engrave"
Subscript	\sub	"Subscript \sub 1"
Superscript	\super	"Superscript \super 2"
Outline	\outl	"\outl Outline"
Shadow	\shad	"\shad Shadow"
Foreground color	\cfn	"\cf1 Foreground1 \cf2 Foreground2 \cf3 Foreground3"
Wave underline	\ulw	"\ulwave Wave underline";
Thick underline	\ulth	"\ulth Thick Underline"
Font size in half points	\fs24	"\fs48 Font Size in half points size 48 will be 24pt in the document"

The syntax starts with the escape character. Next, curly braces enclose the keyword raw, followed by the control words and text to be formatted. This entire specification is enclosed in double quotation marks. In the sample TITLE statement, all you have to do is substitute your control words and text where we show *<string>*. The RTF control words won't change unless Microsoft changes them.

title "^{raw *<string>*}";

SAS 9.1.3 syntax starts with the escape character and the letter "R."

```
title "^R'<string>'";
```

Note that while some control words might be case-insensitive, in our testing we noted that the bullet control word, for example, would not work if you specified \Bullet instead of \bullet (with a lowercase "B").

There is an alternate ESCAPECHAR method that involves using the RTF control word directly without using ^{raw}. This alternate syntax assumes that you are only going to send the RTF control words to an open RTF destination. In this case, all you need to do is set PROTECTSPECIALCHARS= to OFF. This is the method most often used in SAS 9.1.3:

title "^S={protectspecialchars=off}<*string*>";

An additional set of control words {\animtext1-\animtext6} can be used to animate text, as shown in Figure 4.56. However, the animation does not display well as a static screen shot, so the Word view of the document is shown instead of Print Preview mode (in fact, when you print your document, the marching red dashes and other animation cannot be printed).

Figure 4.54: (Print Preview mode) TITLE Statement Output

Figure 4.55: (Print Preview mode) FOOTNOTE Statement Output

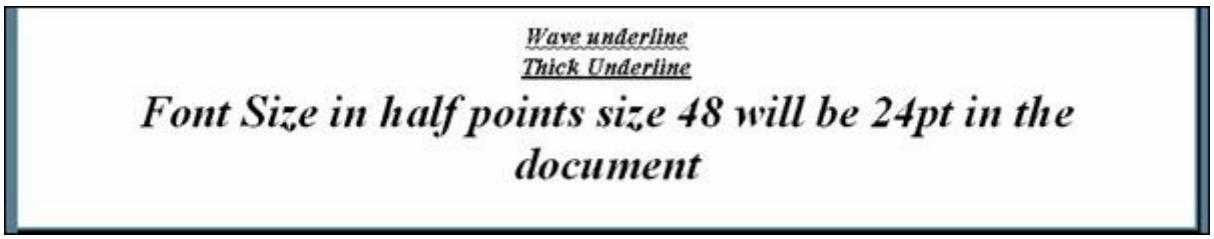


Figure 4.56: Document View in Word

Sending RTF Field Codes and Control Strings Only to RTF Output

The examples in this chapter so far have assumed that you are creating only RTF output. An important feature of ODS, however, is that you can create more than one type of output at a time. Executing your program once, you can create both a page for your Web site and an RTF version of the same report to
e-mail to your team.

Problems can occur if you start using some of the features presented in this chapter with output going to destinations other than RTF, because those destinations cannot correctly interpret the RTF specifications. The following code demonstrates the problem. It creates three output files: an RTF file, an HTML file, and a LISTING file. The TITLE statement specifies custom page numbers using the RTF field code.

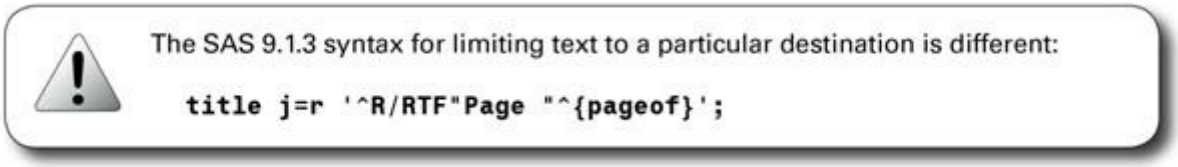
```
      ODS ESCAPECHAR='^';
title j=r 'Page ^{pageof}';
ODS RTF FILE='PageNumRTF.rtf';
ODS HTML FILE='PageNumHTML.html';
      ODS LISTING;
proc print data=clips;
      run;
ODS _ALL_ CLOSE;
```

This code creates page numbers in the “Page X of Y” format in the RTF output, but neither the HTML destination nor the LISTING destination can interpret the escape sequence and RTF field code. The HTML report in Figure 4.57 shows the text “Page” in the title, and it shows that the “^{pageof}” field code is ignored. The title in the LISTING destination report in Figure 4.58 includes both the text “Page” and the uninterpreted RTF specification for the custom page numbers.

Even if you could generate page numbers for the HTML destination, they would be meaningless, because Web pages scroll continuously. A better solution is to block the text “Page” from appearing in the HTML destination. ODS provides a tool for doing this: an escape sequence that limits the following text to a particular destination.

```
title j=r ' ^{dest[rtf] Page ^{pageof}}';
```

When this TITLE statement is used for RTF output, the “Page X of Y” numbers are generated as before. The title is blank in the output sent to the HTML destination, as shown in Figure 4.59. However, the listing output is still a problem, as shown in Figure 4.60.



If you need listing output as well as RTF output, you need a different approach to add the page numbers. You could specify an ODS RTF TEXT statement instead of the TITLE statement. However, this would put the page numbers in the body of the RTF document and not in the Header or Footer area. In this situation, you might need to consider submitting a separate program for the LISTING destination or coding a SAS macro program to conditionally insert

the appropriate words or HTML codes into your output files. These examples cover page numbers generated with escape sequences, but they raise issues that apply to any situation where you are adding special codes, either RTF field codes or other escape sequences.

Figure 4.57

Page

Obs	Tone	Mentions	Compete	Media	Words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Neutral	6	1	Web	1018

Figure 4.58

Output - (Untitled)

Page ^{pageof}

Obs	Tone	Mentions	Compete	Media	Words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Neutral	6	1	Web	1018
4	Positive	5	3	Magazine	561
5	Negative	5	2	Radio	88
6	Positive	1	4	Web	285
7	Positive	2	1	Radio	346

Figure 4.59

Obs	Tone	Mentions	Compete	Media	Words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Neutral	6	1	Web	1018

Figure 4.60

Output - (Untitled)

^{dest[rtf] Page ^{pageof}}

Obs	Tone	Mentions	Compete	Media	Words
1	Negative	2	3	Newspaper	308
2	Negative	1	1	Radio	97
3	Neutral	6	1	Web	1018
4	Positive	5	3	Magazine	561
5	Negative	5	2	Radio	88
6	Positive	1	4	Web	285
7	Positive	2	1	Radio	346

Importing RTF Output into a Slide Show

The previous chapter showed how to import HTML output into a slide show. You can also use RTF output in this way. You might want to experiment with both to see which works better for you.

There are two ways you can get RTF output into PowerPoint. The first option is to open PowerPoint, and then open your RTF file directly. However, if you try this, you will discover that PowerPoint is not very good at reading RTF files. It pulls in your information, but it creates a slide for each title and a slide for each row in the table. You'd have to do a lot of editing if you used this approach.

The better way to import your RTF results into PowerPoint is to first open the RTF file in a word processor. Then select all of the output you want on the first slide, and copy and paste it into a blank slide in PowerPoint. This way your output is recognized as a table by PowerPoint.

Figure 4.61 shows what RTF output looks like when it is first pasted into PowerPoint. The table structure is intact, but the table is too small for the slide, and it's off-center. To fix this, you need to drag the table into the center of the slide. Make sure you select the entire table when you move it. If you are not careful, you might select only the text in one of the table cells.

After you have the table where you want it, and while it's still selected, you can resize it to fit your slide. In this example, the table is small and narrow, and it needs to be resized so that it is larger and wider.

Depending on which word processor you are using and which version of PowerPoint you are using, resizing might affect the entire table, or it might just affect the table grid, but not the text inside. You might find that you need to increase the font size to fit the table.

To fix the fonts, keep the entire table selected, and then use the toolbar button to increase the font size. This increases all of the fonts proportionately.

Make sure you use a blank slide for pasting in your RTF output. Otherwise, PowerPoint adds text boxes for the title and contents of the slide, which will get in your way as you reformat your SAS output. After you have the SAS output set up the way you want, it's easy to add a text box for a title or explanatory text.

After you've centered and resized the table, as in Figure 4.62, your results should look just like they did in your word processor. If you want, you can take the reformatting further, by using a design template to change the background or by applying a new slide layout.

Figure 4.61

Analysis Variable : AnnualSalary Annual Salary		
Department	Mean	Sum
Administration	51014	3571000
Manufacturing	51507	10456000
Marketing	53465	3849500
R&D	53587	2786500

Figure 4.62

Payroll Status Report

Analysis Variable : AnnualSalary Annual Salary		
Department	Mean	Sum
Administration	51014	3571000
Manufacturing	51507	10456000
Marketing	53465	3849500
R&D	53587	2786500

More Enhancement of RTF Output Using RTF Control Words

ODS has several ways to enhance the titles, footnotes, column labels, and even table cells in your RTF output. We have seen how to do this in previous examples in this chapter by using the escape character and STYLE= option. We have also seen usage of the {pageof} function, which the word processor interprets to insert the "Page X of Y" associated in the RTF document. In another example, we've shown only a few of the many RTF control words that you can use to customize the RTF output that ODS creates. The fact is that RTF has many more control words. A careful study of RTF control words will prove rewarding if RTF is your destination of choice.

Table 4.2 presents a short list of some additional RTF control words that can perform special formatting when used in a title, in a footnote, or sometimes in a table cell.

Table 4.2: Selected RTF Control Words for Formatting

Style	RTF Control Word	Action
Underline	\ul \uld	Underline with a regular underline. Underline with a dotted underline.
Font style	\strike \embo	Strike out text. Emboss text.
Highlight	\highlight*n*	Highlight text in a specific color identified by number. There are 16 pre-defined highlight colors that range sequentially from 0 to 16, with value 8 currently unused. The *n* specifies the number associated with a highlight color.
New line	\line	Add title or footnote lines. With this option, you can add more than 10 title or footnote lines to your RTF document, which is the maximum that the SAS TITLE and FOOTNOTE statements allow.

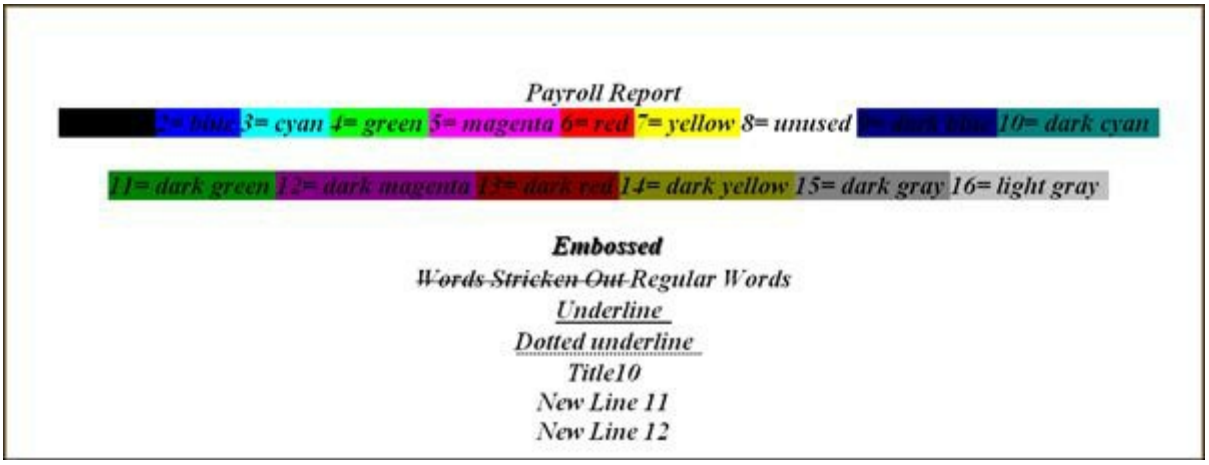
To use these control words, you need to turn off the protection of the special characters such as the backward slash (\) used by RTF. You can either turn off protection of special characters in a style template or by using the ODS ESCAPECHAR and RAW method shown in the code below:

```
ODS RTF FILE='Payroll_Title.rtf';
ODS ESCAPECHAR='^';
title "Payroll Report";
title2 "^{raw \highlight1 1= black \highlight2 2= blue "
      "\highlight3 3= cyan \highlight4 4= green "
      "\highlight5 5= magenta \highlight6 6= red "
      "\highlight7 7= yellow \highlight8 8= unused "
      "\highlight9 9= dark blue \highlight10 10= dark cyan }";
title4 "^{raw \highlight11 11= dark green \highlight12 12= dark magenta "
      "\highlight13 13= dark red \highlight14 14= dark yellow "
      "\highlight15 15= dark gray \highlight16 16= light gray }";
title6 "^{raw  \embo Embossed }";
title7 "^{raw \strike Words Stricken Out \strike0 Regular Words}";
title8 "^{raw  \ul Underline \ul0}";
title9 "^{raw \uld Dotted underline \uld0}";
title10 "^{raw Title10 \line New Line 11 \line New Line 12}";
footnote j=l "Footnote 1";
footnote2 "^{raw This is \ul footnote2 \ul0 with underlining.}";
```

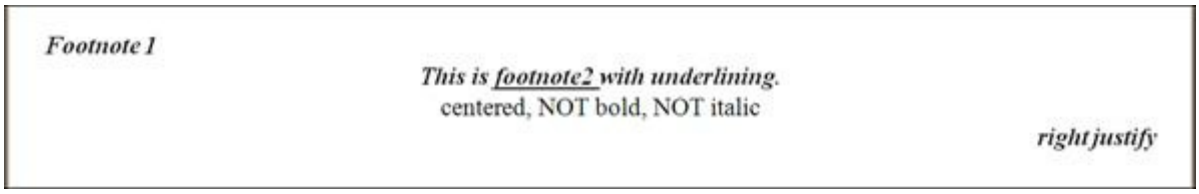
```
footnote3 "^{raw \b0\i0 centered, NOT bold, NOT italic }";
footnote4 j=r "right justify";
proc means data=hr nonobs maxdec=0 mean sum;
   class Department;
   var AnnualSalary;
run;
ODS RTF CLOSE;
```

In Print Preview, as shown in Figure 4.63, you can see the effect of the RTF strings in the titles and the footnotes. The RTF control words used in the FOOTNOTE statement might be of special interest to you if you generate clinical trial tables. To see the style template method for using these same RTF control strings, refer to the style template example in Chapter 18, "Doing More with Styles and ODS."

Figure 4.63 Print Preview Top of Page



Print Preview Bottom of Page



For more information about RTF control words and how to use them with ODS RTF output, search on the support page on the SAS Web site for "Adding RTF Control Words."

The New TAGSETS.RTF Destination

One of the main issues with the original RTF destination was that ODS allowed the word processor (such as Microsoft Word) to control vertical measurement. Because the word processor was in control, if two people opened the same RTF file, they might each get different page counts in the word processor if they changed their margins. There were no hard page breaks inserted in the RTF file except the page break at the start of each procedure.

The TAGSETS.RTF destination is the newest member of the RTF family. It is like the RTF destination in that it creates RTF control strings around your SAS output. The method by which the RTF control strings are created, however, is different from the method used in the original RTF destination.

The original RTF destination produces control strings based on instructions that are "hard-coded" in the SAS executable program. With the new TAGSETS.RTF destination, the instructions that produce the RTF control strings are provided by a tagset template. We discuss tagset templates in more detail in Chapter 12, "Using Tagsets to Generate MARKUP Output."

This program creates a multi-page report in the original RTF destination and in the TAGSETS. RTF destination.

```
    ODS RTF FILE='HRlist.rtf';
ODS TAGSETS.RTF FILE='HRlist_tagsets.rtf';
title 'Human Resources List';
proc print data=hr(obs=75);
run;
ODS _ALL_ CLOSE;
```

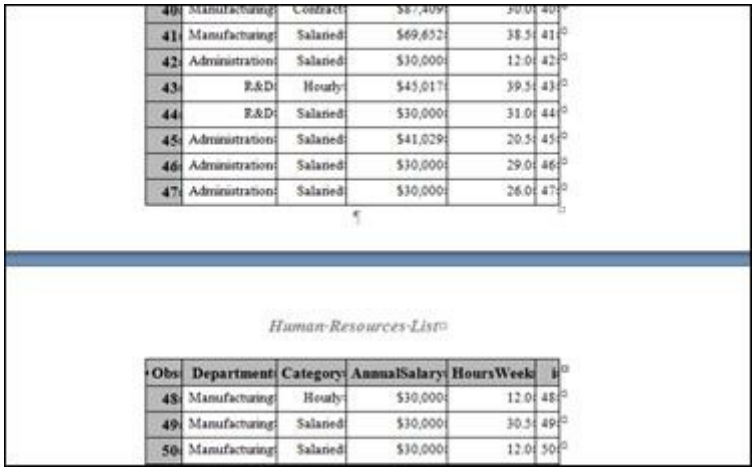
You can see in Figure 4.64 that on the break between page 1 and page 2, there is no indication of the table continuation for the original RTF output file. However, in Figure 4.65, for the TAGSETS.RTF output file, the word "Continued" appears at the break between page 1 and page 2. Notice also the explicit page break between page 1 and page 2 in the TAGSETS.RTF output.

These differences mean that if we change the bottom margin in the files to 2 inches, you will see different behavior in each file. In the TAGSETS.RTF file, the page break would remain at the end of observation 46, exactly where it was in Figure 4.65. However, the page break in the original RTF file would change if the margins changed. The two outputs are placed side by side in Figure 4.66, so you can see the bottom of page 1 for both files.

The TAGSETS.RTF destination might or might not produce the desired behavior for your report. If you like the way that the original RTF destination produces output, then you can stick with the original method. However, if you want SAS to insert explicit page breaks into the RTF file, use TAGSETS.RTF.

Make sure you either issue an ODS _ALL_ CLOSE statement or an ODS TAGSETS.RTF CLOSE statement. An alternate way to invoke this destination is by using the ODS MARKUP statement. This alternate method of invoking markup destinations using the ODS MARKUP statement is discussed in Chapter 12, "Using Tagsets to Generate MARKUP Output."

Figure 4.64: Original RTF Report File

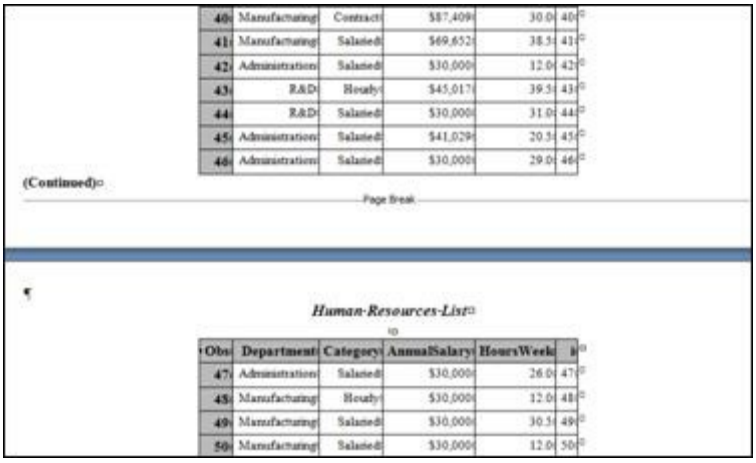


40	Manufacturing	Contract	$87,409	30.0	40
41	Manufacturing	Salaried	$69,652	38.5	41
42	Administration	Salaried	$30,000	12.0	42
43	R&D	Hourly	$45,017	39.5	43
44	R&D	Salaried	$30,000	31.0	44
45	Administration	Salaried	$41,029	20.5	45
46	Administration	Salaried	$30,000	29.0	46
47	Administration	Salaried	$30,000	26.0	47

Human Resources List

Obs	Department	Category	AnnualSalary	HoursWeek	
48	Manufacturing	Hourly	$30,000	12.0	48
49	Manufacturing	Salaried	$30,000	30.5	49
50	Manufacturing	Salaried	$30,000	12.0	50

Figure 4.65: TAGSETS.RTF Report File



40	Manufacturing	Contract	$87,409	30.0	40
41	Manufacturing	Salaried	$69,652	38.5	41
42	Administration	Salaried	$30,000	12.0	42
43	R&D	Hourly	$45,017	39.5	43
44	R&D	Salaried	$30,000	31.0	44
45	Administration	Salaried	$41,029	20.5	45
46	Administration	Salaried	$30,000	29.0	46

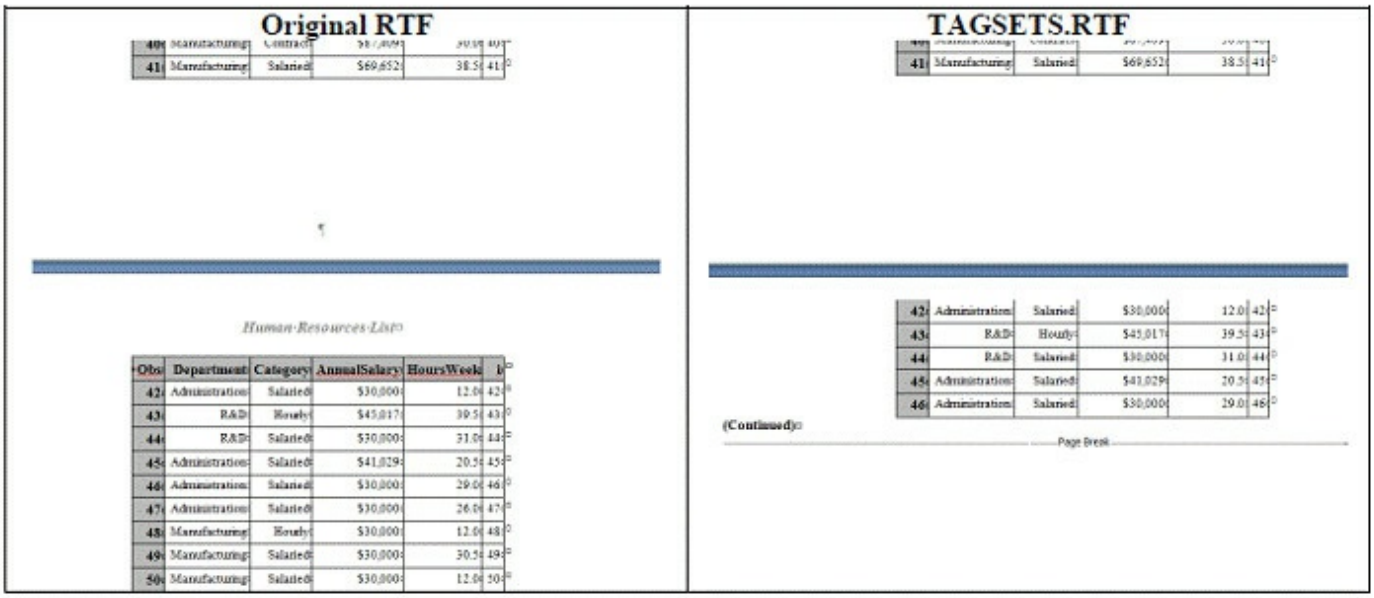
(Continued)

Page Break

Human Resources List

Obs	Department	Category	AnnualSalary	HoursWeek	
47	Administration	Salaried	$30,000	26.0	47
48	Manufacturing	Hourly	$30,000	12.0	48
49	Manufacturing	Salaried	$30,000	30.5	49
50	Manufacturing	Salaried	$30,000	12.0	50

Figure 4.66: Change Bottom Margin to 2 Inches



Inserting Text with the PREPAGE= Option

We have seen already how to insert text into destination output files using the ODS TEXT= option. Another way to insert text above a table on a page is to use the PREPAGE= option.

In the original RTF destination, the PREPAGE= option was used with the WORDSTYLE option to create a custom table of contents (before the CONTENTS= option was available). Consider the following program:

```
ODS RTF FILE='prepage.rtf'
  PREPAGE ='Caption Above Table';

ODS TAGSETS.RTF FILE='prepage_tagsets.rtf'
        PREPAGE='Caption Above Table';

title 'Human Resources Reports';

proc means data=hr nonobs maxdec=0 mean sum;

   class Department;

   var AnnualSalary;

run;

proc print data=hr(obs=25);

run;

ODS _ALL_ CLOSE;
```

If you review the outputs, you will see that the PREPAGE= option text appears at the top of both tables for the original RTF destination, as shown in Figure 4.67. However, the PREPAGE= text appears only above the first table for the TAGSETS.RTF file, as shown in Figure 4.68.

In order to put PREPAGE= text above the PROC PRINT output in the TAGSETS.RTF destination, you would need to add a second ODS TAGSETS.RTF statement to your code preceding the PROC PRINT step. The output is shown in Figure 4.69.

```
ODS TAGSETS.RTF FILE='prepage_tagsets2.rtf'
        PREPAGE='Caption Above Means';

title 'Human Resources Reports';

proc means data=hr nonobs maxdec=0 mean sum;

   class Department;
```

```
  var AnnualSalary;
run;

ODS TAGSETS.RTF PREPAGE='PREPAGE Above Print';
proc print data=hr(obs=25);
run;
ODS _ALL_ CLOSE;
```

So with the PREPAGE= option, you can see that there are minor differences between TAGSETS.RTF and the regular RTF destination. In upcoming topics, we explore options that are unique to the TAGSETS.RTF destination.

Figure 4.67: Original RTF with PREPAGE= Option

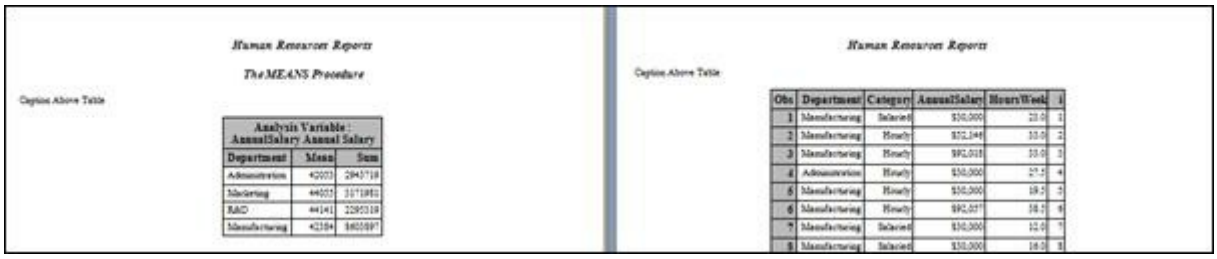


Figure 4.68: TAGSETS.RTF with PREPAGE= Option

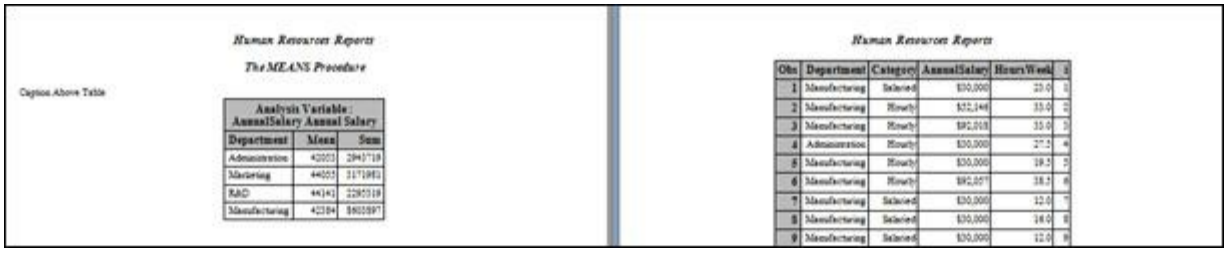


Figure 4.69

Using the TAGSETS.RTF OPTIONS Suboptions

ODS TAGSETS.RTF supports the use of a suboption list in order to provide suboptions that change the way that RTF control strings are written to the RTF output file. The way to see the list of allowable suboptions is to actually use a suboption:

```
ODS TAGSETS.RTF FILE='prepage_suboption.rtf'
  OPTIONS(doc='Help');

ODS _ALL_ CLOSE;
```

When you use the DOC='HELP' suboption, you don't actually need to have any SAS code inside the ODS sandwich. This suboption writes the current Help text to the SAS log.

The entire text of the TAGSETS.RTF Help file is shown in Figure 4.70.

Figure 4.70: TAGSETS.RTF Help File

```
==============================================================================
The RTF Tagset Help Text.

This Tagset/Destination helps with the creation of RTF files
for MS Word

==============================================================================

These are the options currently supported by this tagset.

Sample usage:

ODS TAGSETS.RTF OPTIONS(doc='Quick',
                        CONTENTS='yes');

ODS TAGSETS.RTF OPTIONS(SPECIFIC_OPTION='value');

Debug_Level: No default value.
     Usage: OPTIONS(Debug_Level=1)
     Description:
       Determine what debugging information should be printed
       to the log.  The values expected are numeric and can be used to
       take whatever action is needed. Used in tagsets being debugged,
       but requires a local tagset to be modified.

Doc:  No default value.
     Help: Displays introductory text and options.
     Quick: Displays available options.
     Settings: Displays Current settings.

CONTENTS: No default value.
     Usage: OPTIONS(CONTENTS='yes')
     Description:
       Adds a table of contents page at the top of the file,
       followed by a page break. This must occur before any other output
       in order to have an effect.
       'yes' and 'on' have the same action.
```

(*continued*)

```
SECT: No default value.
     Usage: OPTIONS(SECT='string')
     Description:
       Inserts RTF controls onto the section data specifications.
       ODS tagsets.rtf OPTIONS(sect='string')
       The special string 'NONE' prevents ANY section data from
       being added.
       Assigning 'no' or 'off' resets the option to NULL.

TABLES_OFF: No default value.
     Usage: OPTIONS(TABLES_OFF='string')
     Description:
       When the string names a STYLE type, that style type is
       not inserted into RTF table specs. The text is inserted
       directly into the document.
       To get a list of the style elements written to the SAS log
       use the special qualifier:
       OPTIONS(TABLES_OFF='STYLE_ELEMENTS').
       Assigning 'no' or 'off' resets the option to NULL.
     Example:
       OPTIONS(TABLES_OFF='USERTEXT') will act on the TEXT='string'
       since it uses the USERTEXT style. The text string will
       not be placed in a table. The style name is case sensitive.

TOC_DATA: No default value.
     Usage: OPTIONS(TOC_DATA='on')
     Description:
       Adds table of contents information to the file
       A value of 'off' or 'no' stops the information from  being added.

TROWD: No default value.
     Usage: OPTIONS(TROWD='string')
     Description:
       Inserts raw RTF specifications directly onto the
       table row header descriptions.
       Assigning 'no' or 'off' resets the option to NULL.

TRHDR: No default value.
     Usage: OPTIONS(TRHDR='string')
     Description:
       Inserts raw tablerow RTF specifications directly onto the
       table row header descriptions.
       Assigning 'no' or 'off' resets the option to NULL.

TROWHDRCELL: No default value.
     Usage: OPTIONS(TROWHDRCELL='string')
     Description:
       Inserts raw text on the trow cells.
       Assigning 'no' or 'off' resets the option to NULL.

==============================================================================
```

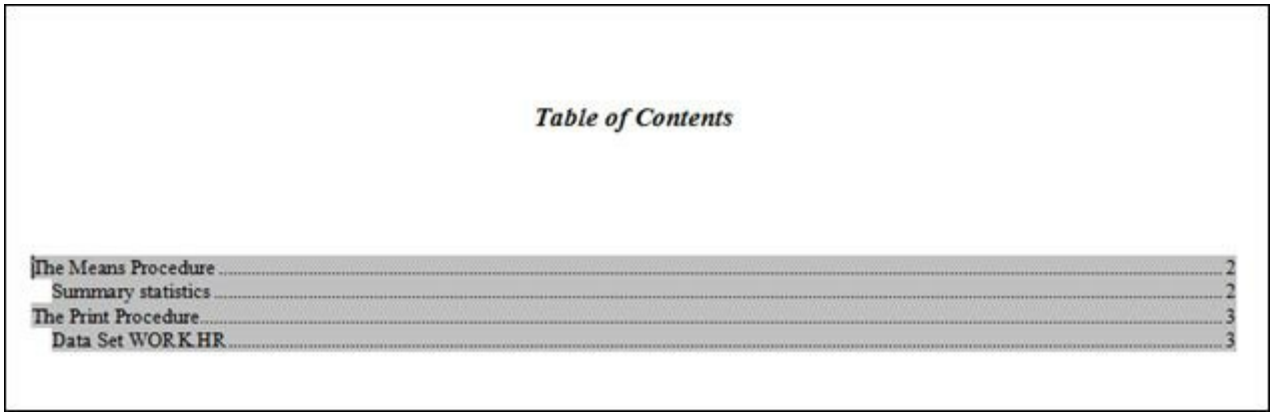
Generating a Table of Contents with TAGSETS.RTF

We have already seen the use of the CONTENTS= option with ODS RTF. To generate a table of contents with TAGSETS.RTF, the method is very similar. Consider the following code:

```
ODS TAGSETS.RTF FILE='TOC_SUBOPTIONS.rtf'
        OPTIONS(CONTENTS='yes' TOC_DATA='yes');

title 'Human Resources Reports';

proc means data=hr nonobs maxdec=0 mean sum;

   class Department;

   var AnnualSalary;

run;

proc print data=hr(obs=25);

run;

ODS _ALL_ CLOSE;
```

Although you might need to right-click farther down on the page than with traditional ODS RTF TOC generation, after you right-click and select Update Field, you will see a table of contents generated as shown in Figure 4.71.

Figure 4.71



Changing Table Headers with RTF Control Strings

If you know how to use RTF control strings, there is even more you can do with suboptions for the TAGSETS.RTF destination. Several suboptions are designed to pass RTF control strings into table headers. These are the TRHDR, TROWD, and TROWHDRCELL suboptions that we feature in this example and the next example.

The TROWD suboption inserts RTF control strings directly into the table row header control strings that format the header cells. The second ODS TAGSETS.RTF statement below places the \RTLROW control string in the suboption list supplied to TROWD.

Figure 4.72 shows the default output from sending the first PROC FREQ step to the TAGSETS.RTF destination. Figure 4.73 shows how the \RTLROW control string specified in the second ODS TAGSETS.RTF statement changes the columns from their default left-to-right order to a right-to-left order.

```
ODS TAGSETS.RTF FILE='TOC_CONTROL.rtf';

proc freq data=hr;
   title 'Show Default Order of Columns';
   tables Department / nocum;
run;

ODS _ALL_ CLOSE;

ODS TAGSETS.RTF FILE='TOC_CONTROL2.rtf' OPTIONS(TROWD="\rtlrow");
proc freq data=hr;
   title 'Show Impact of RTLROW Control String';
   tables Department / nocum;
run;

ODS _ALL_ CLOSE;
```

If you run the same PROC FREQ code and replace the TROWD suboption with the TROWHDRCELL suboption, you can pass a control string into the \TROW (table row) cells. This location in the RTF file allows you to provide settings for every cell in the table row.

```
ODS TAGSETS.RTF FILE='TOC_CONTROL3.rtf'
    OPTIONS(TROWHDRCELL="\clbghoriz\clcfpat3");
proc freq data=hr;
```

```
    title 'Show Impact of CLBGHORIZ and CLCFPAT3 Control Strings';
    tables Department / nocum;
  run;

  ODS _ALL_ CLOSE;
```

In this example, the \CLBGHORIZ control string places horizontal lines in the table headers. Then, the \CLCFPAT3 control string assigns a color to the pattern that has been placed in the table headers. In this example, we use color number '3', which is cyan, as shown in Figure 4.74.

Figure 4.72

Show Default Order of Columns

Department		
Department	Frequency	Percent
Administration	70	17.63
Marketing	72	18.14
R&D	52	13.10
Manufacturing	203	51.13

Figure 4.73

Show Impact of RTLROW Control String

Department		
Percent	Frequency	Department
17.63	70	Administration
18.14	72	Marketing
13.10	52	R&D
51.13	203	Manufacturing

Figure 4.74

Show Impact of CLBGHORIZ and CLCFPAT3 Control Strings

Department		
Department	Frequency	Percent
Administration	70	17.63
Marketing	72	18.14
R&D	52	13.10
Manufacturing	203	51.13

Rotating or Aligning Table Headers with RTF Control Strings

With a different RTF control string, you can rotate the table headers so that they appear vertically in the table, or you can use RTF control strings to alter the alignment of text in a cell. Of course, if you're going to rotate the headers, you need to increase the row height. There's an RTF control string to increase the row height, but it has to be provided as part of the RTF header description.

The TRHDR suboption passes an RTF control string into this header location. So you need to use the TRHDR suboption to increase the row height, and then in the TROWHDRCELL suboption, you provide the appropriate control string to alter the headers.

```
ODS TAGSETS.RTF FILE='HEADERS_UP.rtf' style=journal
   OPTIONS (TRHDR="\trrh1225" TROWHDRCELL="\cltxbtlr");
proc print data=hr(obs=25);
  title 'Using TRRH1225 and CLTXBTLR Control Strings';
run;
ODS _ALL_ CLOSE;

ODS TAGSETS.RTF FILE='HEADERS_DOWN.rtf' style=journal
   OPTIONS (TRHDR="\trrh1225" TROWHDRCELL="\cltxtbrl");
proc print data=hr(obs=25);
  title 'Using TRRH1225 and CLTXTBRL Control Strings';
run;
ODS _ALL_ CLOSE;
```

In the preceding code, we've used the JOURNAL style to remove the interior lines from the table and header cells in order to emphasize the rotation. The \TRRH1225 control string specifies a height for the row header in a measurement called "twips" (twentieths of a printer's point, where 1 twip is approximately 1/1440 inch). Then the \CLTXBTLR and \CLTXTBRL control strings control differing placements of the header strings.

With the \CLTXBTLR control string, you can see in Figure 4.75 that the headers go up. Then in Figure 4.76, the \CLTXTBRL control string causes the header strings to go down.

The RTF specification also includes several alignment control strings. This last example uses a simple alignment string with the same PROC PRINT code as that above. Figure 4.77 shows the result of using the \CLVERTALC control string in the TROWHDRCELL suboption.

```
ODS TAGSETS.RTF FILE='HEADERS_CTR.rtf' style=journal
```

```
    OPTIONS (TRHDR="\trrh1225" TROWHDRCELL="\clvertalc");
proc print data=hr(obs=25);
  title 'Using TRRH1225 and CLVERTALC Control Strings';
run;
ODS _ALL_ CLOSE;
```

As you can see, the header text is centered vertically in the enlarged row height. The TAGSETS.RTF template, which controls output creation, does not check the validity of the strings that you pass to the output file. Therefore, study of the RTF specification is essential to successful output file creation if you intend to use these suboptions.

Figure 4.75

Using TRRH1225 and CLTXBTLR Control Strings

Obs	Department	Category	AnnualSalary	HoursWeek
1	Manufacturing	Salaried	$30,000	23.0
2	Manufacturing	Hourly	$52,146	33.0
3	Manufacturing	Hourly	$92,018	33.0

Figure 4.76

Using TRRH1225 and CLTXTBRL Control Strings

Obs	Department	Category	AnnualSalary	HoursWeek
1	Manufacturing	Salaried	$30,000	23.0
2	Manufacturing	Hourly	$52,146	33.0
3	Manufacturing	Hourly	$92,018	33.0

Figure 4.77

Using TRRH1225 and CLVERTALC Control Strings

Obs	*Department*	*Category*	*AnnualSalary*	*HoursWeek*
1	Manufacturing	Salaried	$30,000	23.0
2	Manufacturing	Hourly	$52,146	33.0
3	Manufacturing	Hourly	$92,018	33.0

Adding Hyperlinks to TAGSETS.RTF Output

Earlier in the chapter, you saw the use of the original ODS ESCAPECHAR syntax to set a hyperlink. With TAGSETS.RTF, you can use the SAS 9.2 form of the ODS ESCAPECHAR syntax to add a hyperlink to your output.

If the escape character has been defined as the caret (^), then the new function call form of the ODS ESCAPECHAR syntax is as follows:

```
^{style[URL="http://www.myweb.com"]Text to Click On}
```

This code creates a hyperlink from the value of the URL= attribute value. The URL= attribute is contained within square brackets, and the text string that would be clickable is specified outside the square brackets, but inside the curly braces.

The following example adds a hyperlink at the bottom of the output using the ODS TAGSETS.RTF TEXT= statement. Rather than putting the hyperlink in a footnote, which would be at the bottom of the page, the TEXT= feature places the hyperlink in a text box right below the procedure output. Remember that if we use the ODS TEXT= form of the option, the text is written to all open destinations.

```
    ODS ESCAPECHAR='^';
ODS TAGSETS.RTF FILE='Hyperlink.rtf';
proc freq data=clips;
      table compete;
    run;
ODS TAGSETS.RTF TEXT='^{style[URL="http://www.myweb.com"]Source: www.myweb.com'};
ODS _ALL_ CLOSE;
```

You can see the text for the hyperlink below the table. This hyperlink is essentially the same as the hyperlink set using the original ESCAPECHAR syntax in the earlier example. As before, when you move the mouse over the text, you can see in Figure 4.78 that it is a hyperlink.

To change the text color of the hyperlink to blue, the updated escape sequence would set the foreground color to blue for the clickable text.

```
    ODS TAGSETS.RTF TEXT='Source: ^{style[URL="http://www.myweb.com"
FOREGROUND=blue]www.myweb.com}';
```

The revised output is shown in Figure 4.79. Now the Web site address looks like a standard hyperlink because it appears in blue, which should prompt the user to click on the link. Note that for this output the word ‘Source:’ is displayed in the default style template color (black), whereas the clickable text is in blue.

Figure 4.78

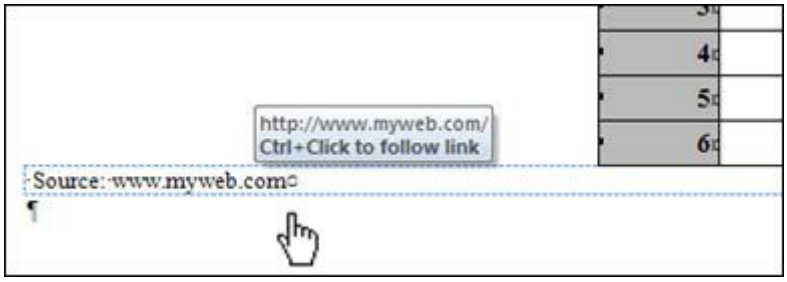
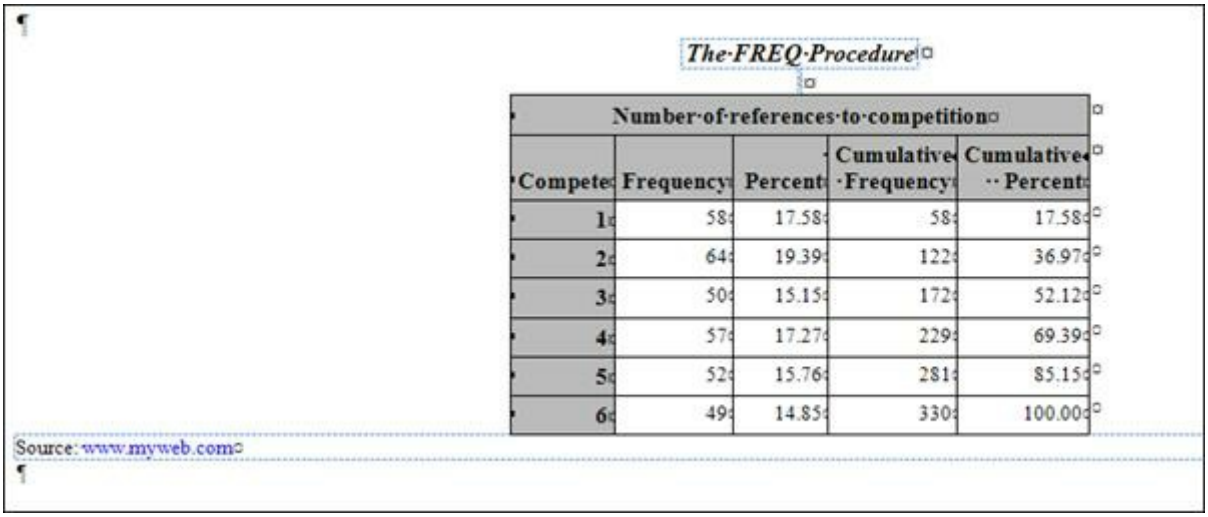


Figure 4.79



The FREQ Procedure

Number of references to competition				
Compete	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	58	17.58	58	17.58
2	64	19.39	122	36.97
3	50	15.15	172	52.12
4	57	17.27	229	69.39
5	52	15.76	281	85.15
6	49	14.85	330	100.00

Source: www.myweb.com

Chapter 5
PDF Output

The previous two chapters focused on ODS output formats that can be displayed and edited electronically. If we all worked in the so-called “paperless office” of the future, we wouldn’t need any other output formats.

However, in the real world, most of us still deliver a large percentage of our results on paper, or people print a report as soon as they receive their electronic copy. If you need to generate your output so that it can be printed, the fastest way to get your SAS results is to send the output directly to a printer or to create a PDF file for e-mail or Web delivery.

ODS Printer family output, which includes PDF, enables you to create attractive black and white or color output. This output has many of the features of HTML output, but with proper page breaks. The tables and fonts look just as good as the RTF output, but you don’t need to open up a word processor to view it. You can create a PDF, PostScript, or PCL file to print later, or you can print the output directly to your printer.

However, there are some issues to keep in mind with ODS Printer family output. Although PDF, PostScript, and PCL are standard file formats, printer definitions and commands vary widely across operating systems. If you’re not in a Windows environment, you might have to invest a little time to learn how best to handle ODS Printer output on your system. This chapter primarily covers ODS PDF, but also covers some of the operating system differences if you do need to use the ODS PRINTER destination.

This chapter uses the data sets CLIPS, COMPLAINTS, GALLERY, and UNIFORMS to illustrate the concepts. These data sets are available on the author pages for this book at http://support.sas.com/authors.

Creating a PDF File

With ODS Printer output your results are either printed output or a file of some type. If you're sharing your results at a meeting in your office, a paper copy might be fine. However, sometimes you do not have a printer available at the time you run your code, or you might want to share your results electronically. In those cases, you need a different option. Otherwise, if you want the same output again, you have to rerun the SAS program that created it.

To create a more permanent version of your output, you can ask ODS to create a PDF output file that you can then print at your convenience. PDF is a proprietary format created by Adobe. It allows other users to see your printed output exactly as you see it, even if they don't have the same printer. They don't have to print it unless they want a copy. After users have downloaded the free Adobe Reader, they can review your results on-screen.

Creating PDF files not only enables you to create very attractive output using ODS, but it also enables you to send the file electronically. This is a perfect solution when you have information you expect others will want to print. ODS HTML output is easily shared electronically, but it is difficult to print because page breaks are controlled by the browser.

Creating PDF output is very simple. Just use ODS PDF as the destination. You will also want to use a FILE= option to name the resulting file.

```
ODS PDF FILE='myfile.pdf';
```

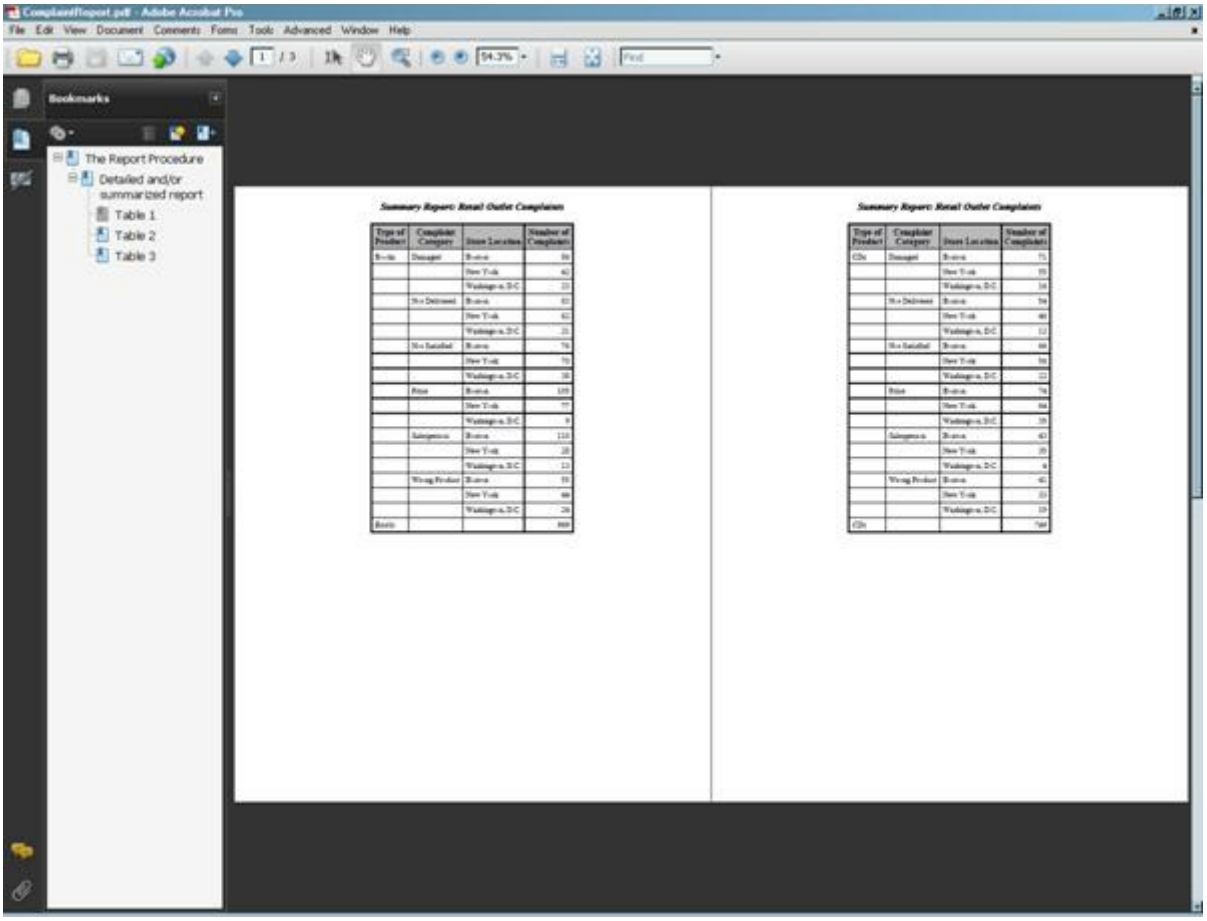
The following code routes the results of a PROC REPORT step to a file called ComplaintReport.pdf.

```
ODS PDF FILE='ComplaintReport.pdf';
proc report data=Complaints nowd;
  column Product Complaint Location NumComplaints;
  define Product / group order=formatted;
  define Complaint / group order=formatted;
  define Location / group order=formatted;
  define NumComplaints / analysis sum;
  break after Product / page summarize;
run;
ODS PDF CLOSE;
```

The results are shown in Figure 5.1. It is displayed using Adobe Acrobat 9 software. The first two pages are shown in "Two-Up" view. Notice how the page has a proper page break at the end of the first table. If this output were created using ODS HTML, the tables would have been separated by horizontal rules, which might or might not have printed as you wanted. If page breaks are important, then PDF is the electronic document format you should use.

The Adobe Reader software is available for a number of platforms (Windows, Macintosh, UNIX) and can be downloaded free of charge at www.adobe.com.

Figure 5.1


ComplaintReport.pdf - Adobe Acrobat Pro
File Edit View Document Comments Forms Tools Advanced Window Help
Bookmarks
The Report Procedure
Detailed and/or summarized report
Table 1
Table 2
Table 3
Summary Report: Retail Outlet Complaints
Summary Report: Retail Outlet Complaints

Creating Bookmarks in Your PDF File

With ODS HTML output, you can generate a table of contents to help you find the results you want in a lengthy report. You also get a table of contents with ODS PDF output.

The PDF format supports the creation of bookmarks, which are displayed by Adobe Reader in a window to the left of the main PDF file. By clicking on a bookmark, you can jump directly to the correct output page. The PDF bookmarks created by ODS are basically the same as the headings and subheadings created in an ODS HTML table of contents.

The following code illustrates this feature. It produces a two-page output file with several tables, which are bookmarked in the PDF output.

```
      title 'Analysis of Consumer Complaints';
ODS PDF FILE='ComplaintAnalysis.pdf';
proc glm data=Complaints;
        class product location;
  model NumComplaints=Product Location;
run;
ODS PDF CLOSE;
```

The resulting PDF file is shown in Figure 5.2. Notice the Bookmarks panel on the left. It displays a tree view of the structure of the output file. It has two top-level bookmarks: one for the “Data” component of the output and one for the “Analysis of Variance” component of the output. Below each of these bookmarks are links to the tables within each component. If you click one of the links, the page with that section or table is displayed in the window on the right.

If the bookmarks do not appear when you open the PDF file, you might need to click the bookmarks icon or bookmarks arrow to display them. Figure 5.2 is shown in Adobe Acrobat 9. Your version of Adobe might have different options to display bookmarks.

As with HTML output, if you use the ODS PROCLABEL statement to assign a new label to the procedure, that label is displayed in the bookmarks. For example, the same text used in the TITLE statement of our example could also be used to generate a procedure label by adding the following ODS PROCLABEL statement to the preceding program:

```
ODS PROCLABEL 'Analysis of Consumer Complaints';
```

Figure 5.3 shows the new results. Notice how the top label has changed from “The GLM Procedure” to “Analysis of Consumer Complaints.”

Figure 5.2

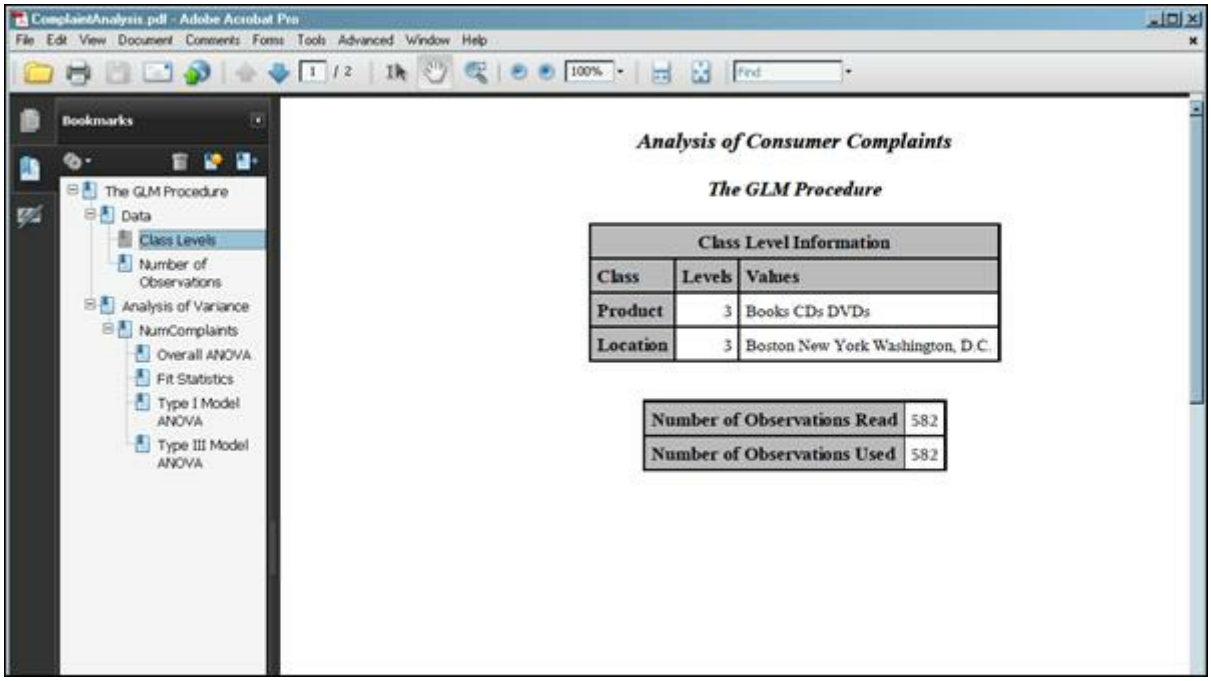
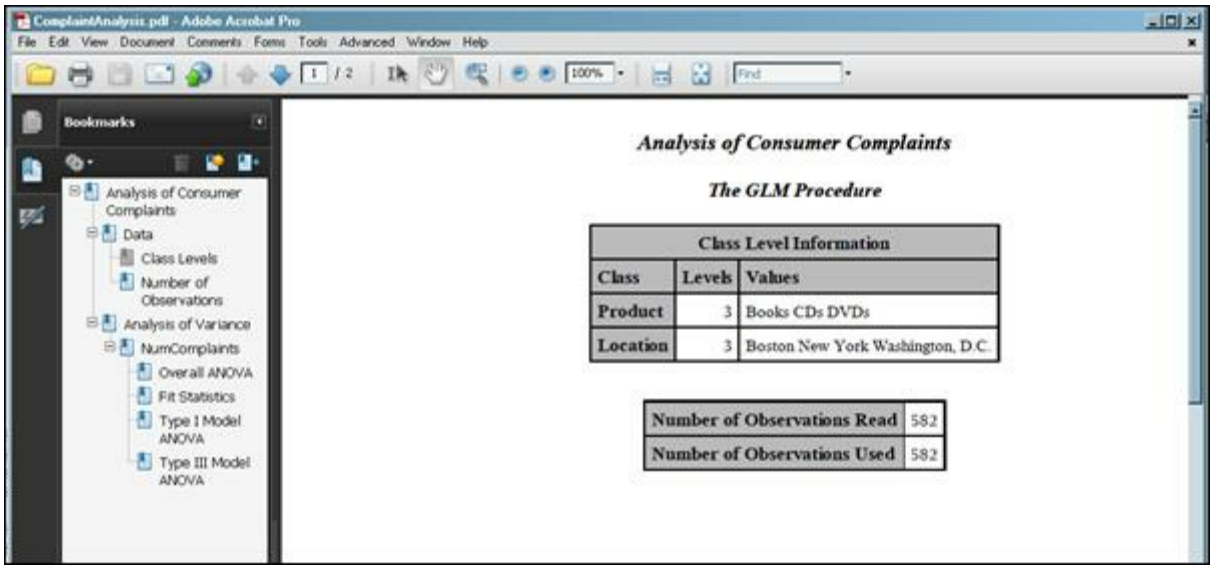

ComplaintAnalysis.pdf - Adobe Acrobat Pro
File Edit View Document Comments Forms Tools Advanced Window Help
Bookmarks
The GLM Procedure
Data
Class Levels
Number of Observations
Analysis of Variance
NumComplaints
Overall ANOVA
Fit Statistics
Type I Model ANOVA
Type III Model ANOVA
Analysis of Consumer Complaints
The GLM Procedure
Class Level Information
Class | Levels | Values
Product | 3 | Books CDs DVDs
Location | 3 | Boston New York Washington, D.C.
Number of Observations Read | 582
Number of Observations Used | 582

Figure 5.3


ComplaintAnalysis.pdf - Adobe Acrobat Pro
File Edit View Document Comments Forms Tools Advanced Window Help
Bookmarks
Analysis of Consumer Complaints
Data
Class Levels
Number of Observations
Analysis of Variance
NumComplaints
Overall ANOVA
Fit Statistics
Type I Model ANOVA
Type III Model ANOVA
Analysis of Consumer Complaints
The GLM Procedure
Class Level Information
Class | Levels | Values
Product | 3 | Books CDs DVDs
Location | 3 | Boston New York Washington, D.C.
Number of Observations Read | 582
Number of Observations Used | 582

Modifying the Display of Bookmarks in Your PDF File

SAS®9 includes many options that can control the display of the contents list in the PDF file that ODS creates. ODS PDF statement options CONTENTS=, BOOKMARKGEN=, BOOKMARKLIST=, and PDFTOC= affect how the table of contents or bookmarks appear in your PDF document.

Specifying the CONTENTS=YES option in your ODS PDF statement generates a separate "Table of Contents" page in your document. Adding this option to the ODS PDF statement for the preceding program generates a "Table of Contents" page as shown in Figure 5.4.

Options BOOKMARKGEN= and BOOKMARKLIST= can control whether bookmarks are generated and displayed. For example, when you set BOOKMARKGEN=NO or BOOKMARKLIST=NONE, bookmarks are not created, as shown by the empty bookmark pane in Figure 5.5. When you set BOOKMARKLIST=HIDE, the PDF bookmarks are still generated internally, but they are not displayed when the file is opened, as shown in Figure 5.6. By default, the BOOKMARKGEN= option is set to YES, so assume that it was in effect when Figure 5.6 was created.

The ODS PDF statement option PDFTOC=, which is new in SAS 9.2, controls the expansion level of the bookmark area when the PDF file is opened. Figure 5.7 shows the PDF file created for the preceding program when PDFTOC=1 is added to the ODS PDF statement. Only the highest level bookmark, "The GLM Procedure," is shown when the file is opened.

Only the BOOKMARKGEN= and BOOKMARKLIST= options that produce desirable results have been shown. Since the default value of BOOKMARKGEN= is YES and the default value of BOOKMARKLIST= is SHOW, you do not have to worry about adverse interactions if you choose these options. Be aware, however, that some combinations of the BOOKMARKGEN= and BOOKMARKLIST= options can cause the bookmark icon to disappear entirely from Adobe Reader 8. This behavior might be different in other versions of Adobe Reader.

Table 5.1 lists combinations of BOOKMARKGEN= and BOOKMARKLIST= and how ODS processes them.

Table 5.1: BOOKMARKGEN= and BOOKMARKLIST= Option Combinations and ODS Results

BOOKMARKGEN= Value	BOOKMARKLIST= Value	ODS Action	Adobe Reader 8 Behavior on Initial File Open
YES	SHOW	Generates list of bookmarks	Shows bookmark list
	HIDE	Generates list of bookmarks	Hides bookmark list
	NO	Does not generate list of bookmarks	In Adobe Reader 8, suppresses bookmark icon entirely
NO	SHOW | HIDE | NO	Does not generate list of bookmarks	In Adobe Reader 8, suppresses bookmark icon entirely

Figure 5.4: CONTENTS=YES Option on ODS PDF Statement

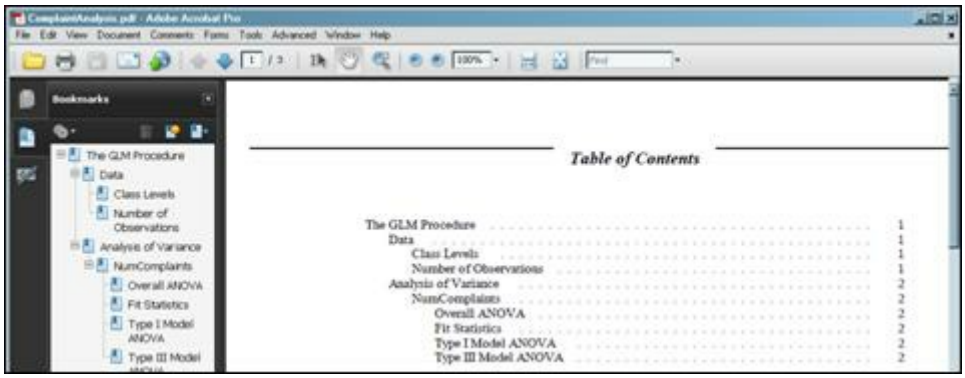


Figure 5.5: BOOKMARKGEN=NO on ODS PDF Statement

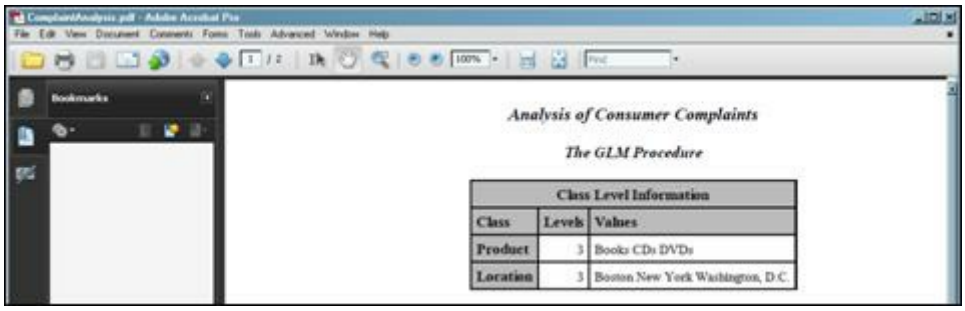


Figure 5.6: BOOKMARKLIST=HIDE on ODS PDF Statement

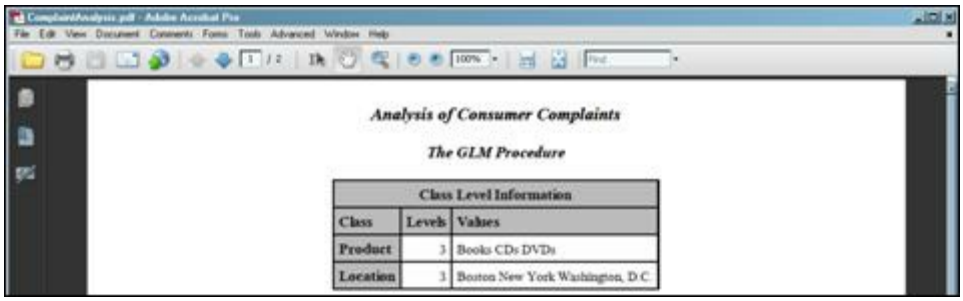
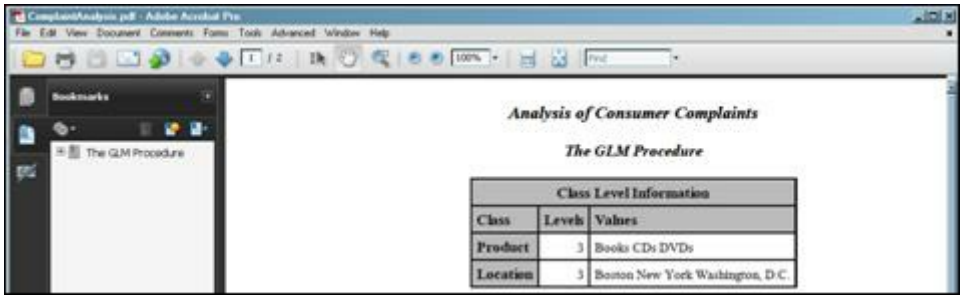


Figure 5.7: PDFTOC=1 on ODS PDF Statement



Creating PDF Files That Include Hyperlinks and Annotations

The PDF file format is a lot more than just a convenient way to share electronic documents. ODS can also add interactive features to your PDF files. You can add a hyperlink to a PDF document, enabling the viewer to link to a related Web site. We have already seen the use of LINK= in the SAS TITLE statement. That feature of ODS works with PDF in the same manner as it works for HTML and RTF. So here we illustrate another method of performing linking and adding annotations to PDF files.

You can add annotations that create pop-up notes with added details. To access these features, you use the STYLE= option and the URL= and FLYOVER= attributes. The STYLE= option is currently supported for the PRINT, REPORT, and TABULATE procedures. In this example, hyperlinks are added to two column headings, and an annotation is added to a third column heading. The hyperlinks enable the report to be linked to Web sites with information about the artists and media described in the table. The annotation explains that the prices are in U.S. dollars. The code is shown below.

```
    ODS PDF FILE='AnnotatedReport.pdf';
proc report data=gallery nowd;
  column artist medium price;
  define artist / group STYLE(Header)=
              {URL="http://www.artistlookupwebsite.com/"};
  define medium / group STYLE(Header)=
              {URL="http://www.artistmediumlookup.com"};
  define price / analysis mean STYLE(Header)=
             {FLYOVER="Prices listed in US$"};
run;
ODS PDF CLOSE;
```

The STYLE= option is applied to each column separately, because each column has a different link or flyover note. In the first two columns, the URL= attribute specifies a link to a related Web site. In the third column, the FLYOVER= attribute specifies the text for the annotation.

The resulting PDF is shown in Figure 5.8. The first two column headings are surrounded by blue boxes, indicating that they are hyperlinks. Clicking on or in a blue-outlined box links the viewer to the appropriate Web site. The icon in the third column indicates that it has an annotation. Double-clicking the icon brings up the note, as shown in Figure 5.9.

The ODS PDF statement option PDFNOTE= controls whether FLYOVER= settings are translated into annotations. Be sure you have not set PDFNOTE=NO or set NOPDFNOTE in your ODS PDF statement because if you do, the annotation technique described above will not work. However, if you're creating both HTML and PDF, and you don't want your HTML flyovers to turn into PDF annotations, then use NOPDFNOTE (or PDFNOTE=NO) to turn them off.

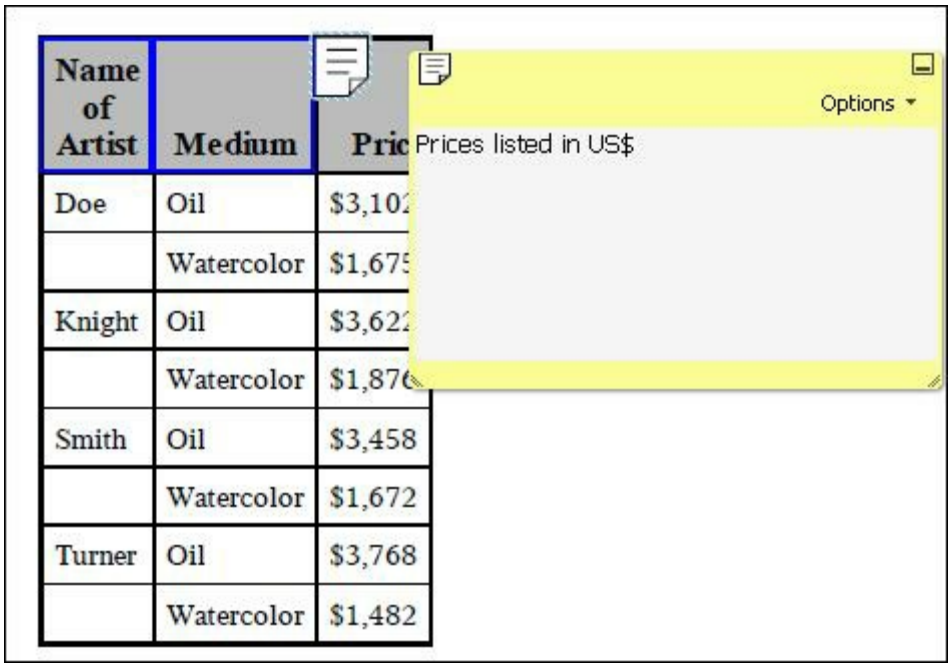
This technique is shown with the ODS PDF destination. If you prefer to distill your own PostScript files, you can use the same technique with ODS PS. However, in that situation you need to

add the PDFMARK option to the ODS PS statement.

Figure 5.8

Name of Artist	Medium	Price
Doe	Oil	$3,102
	Watercolor	$1,675
Knight	Oil	$3,622
	Watercolor	$1,876
Smith	Oil	$3,458
	Watercolor	$1,672
Turner	Oil	$3,768
	Watercolor	$1,482

Figure 5.9



Another Means of Adding Hyperlinks and Annotations to PDF Output

The previous technique works great if you want to add hyperlinks or annotations to PDF output from the PRINT, REPORT, or TABULATE procedures. However, if you want to use a different procedure, you don't have access to the STYLE= option (except in a modified Table template), so you can't apply the URL= or FLYOVER= attributes. Also, you might want to put a hyperlink or annotation somewhere else in your output, not just in the table.

Another way to control style attributes is to use the escape sequence ^{STYLE} method (new for SAS 9.2). You can use an escape sequence almost anywhere in your output where you can add text: titles, footnotes, labels, etc. The following example uses an escape sequence to create a hyperlink on one of the words in the table footnote, as shown in Figure 5.10.

```
ODS ESCAPECHAR='^';
ODS PDF FILE='HyperlinkedReport.pdf';
footnote '^{style[URL="http://threeartgalleries.info"]Click for more gallery info}';
proc tabulate data=gallery;
  class gallery;
  var price;
  tables gallery,price*(n*f=5. (mean min max)*f=dollar8.);
run;
ODS PDF CLOSE;
```

This could also be done using the TEXT= option in an ODS PDF statement. This statement would be inserted into the code following the PROC UNIVARIATE step, and the FOOTNOTE statement would be omitted from the program. This approach has the added advantage of creating a footnote much closer to the table itself, as shown in Figure 5.11. The program that produced Figure 5.10 placed the footnote at the bottom of the page, leaving a lot of empty space between the table and the footnote.

```
ODS PDF TEXT= '^{style[just=c URL="http://threeartgalleries.info"]Click for more gallery info}';
```

Annotations can be added to titles, footnotes, or text in the same way by using the FLYOVER= option. Hyperlinks and annotations can also be embedded in format values. The following PROC FORMAT code creates annotations for each value of the variable GALLERY:

```
proc format;
  value $galft
      'East Side'='^{style[FLYOVER="130 East 23rd"]East side}'
      'Downtown'='^{style[FLYOVER="10 Elm Street"]Downtown}'
      'Mall'='^{style[FLYOVER="Southside Mall Blvd"]Mall}';
run;
```

Applying the format $GALFT to the variable GALLERY generates annotations in the PDF output with the address for each gallery location, as shown in Figure 5.12. The same technique can add annotations to variable labels and values of character variables.

Figure 5.10

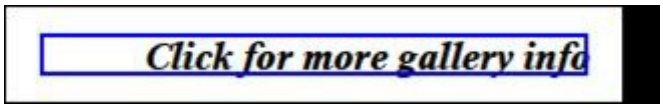
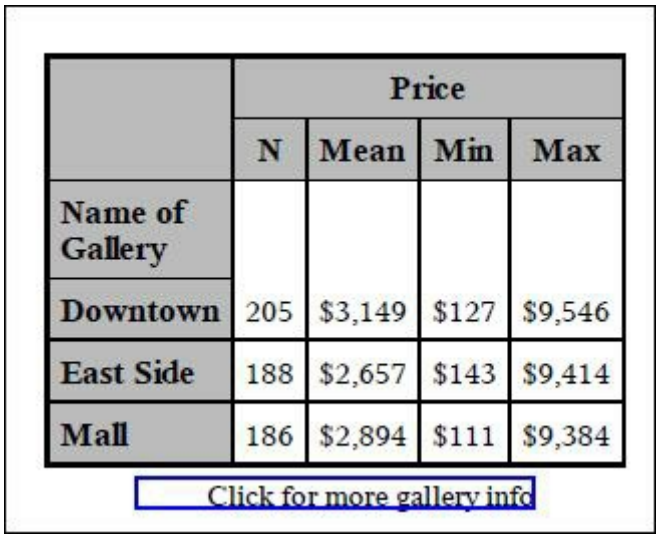


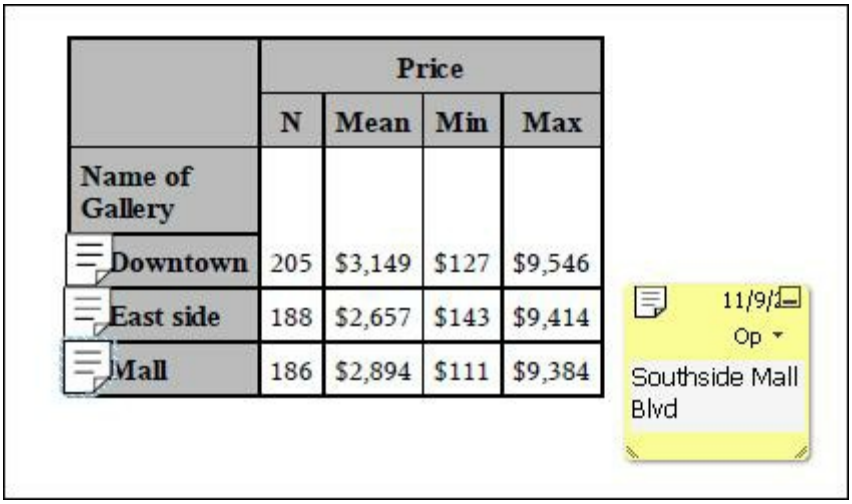
Figure 5.11



	Price			
	N	Mean	Min	Max
Name of Gallery				
Downtown	205	$3,149	$127	$9,546
East Side	188	$2,657	$143	$9,414
Mall	186	$2,894	$111	$9,384

Click for more gallery info

Figure 5.12



	Price			
	N	Mean	Min	Max
Name of Gallery				
Downtown	205	$3,149	$127	$9,546
East side	188	$2,657	$143	$9,414
Mall	186	$2,894	$111	$9,384

This technique does not work for all text and all output locations. Sometimes the procedure itself manipulates the text and the special formatting is lost. The best way to figure out what works is trial and error.

Optimizing PDFs to Reduce Size and Increase Compatibility

PDF files that are created by ODS in SAS®9 have a compression level set automatically. The compression levels range from 0 (no compression) to 9 (highest compression). The default compression level is set to 6. The following ODS PDF statements are equivalent:

```
ODS PDF FILE='cmprsdef.pdf';
ODS PDF FILE='cmprs6.pdf' compress=6;
```

Rather than focus on the SAS code here, what we highlight instead is a benchmarking technique that enables you to determine the best compression level for your PDF output. We chose a simple PROC PRINT of SASHELP.SHOES for this test. When you do your benchmarking, you will need to put your procedure(s) of choice inside the ODS PDF statements and remove the PROC PRINT step.

Macro program TESTCP submits a PROC PRINT of SASHELP.SHOES and writes the output to a PDF file. Macro parameter CNUM is the compression level. Its default value is 0 for no compression. Macro parameter CNUM is specified as the value of the COMPRESS= option in the ODS PDF statement. Macro program TESTCP is called ten times, and it creates one PDF file each time it executes. The DATA step at the bottom of the program reads the directory information for the ten PDF files and writes the information to the SAS log.

```
%macro testcp(cnum=0);
  ODS PDF FILE="cmprs&cnum..pdf" COMPRESS=&cnum;
  proc print data=sashelp.shoes;
  run;
  ODS PDF CLOSE;
%mend testcp;
%testcp()
%testcp(cnum=1)
%testcp(cnum=2)
%testcp(cnum=3)
%testcp(cnum=4)
%testcp(cnum=5)
%testcp(cnum=6)
%testcp(cnum=7)
%testcp(cnum=8)
%testcp(cnum=9)
filename dirinfo pipe 'dir cmprs*.pdf ';
data _null_;
  length infoline $200;
  infile dirinfo truncover length=lg;
  input infoline $varying. lg;
  putlog _infile_;
run;
```

The directory information from the SAS log, as shown in Figure 5.13, shows the results for this set of tests. The highest compression benefit came from changing from compression level 0 (774,317 bytes) to compression level 1 (180,276 bytes). So in this instance, the default compression level of 6 would have been adequate. Of course, your “mileage may vary” as you determine your own correct compression level.

Figure 5.13

```
774,317 cmprs0.pdf
180,276 cmprs1.pdf
176,358 cmprs2.pdf
172,157 cmprs3.pdf
170,883 cmprs4.pdf
163,433 cmprs5.pdf
161,281 cmprs6.pdf
160,625 cmprs7.pdf
159,835 cmprs8.pdf
159,743 cmprs9.pdf
```

The COMPRESS= option works only for invocations of ODS PDF or ODS PRINTER with the PDF option. The option is inappropriate for ODS POSTSCRIPT or ODS PRINTER without the PDF option, because compression is a PDF-specific capability.

Creating PDFs with Customized Metadata

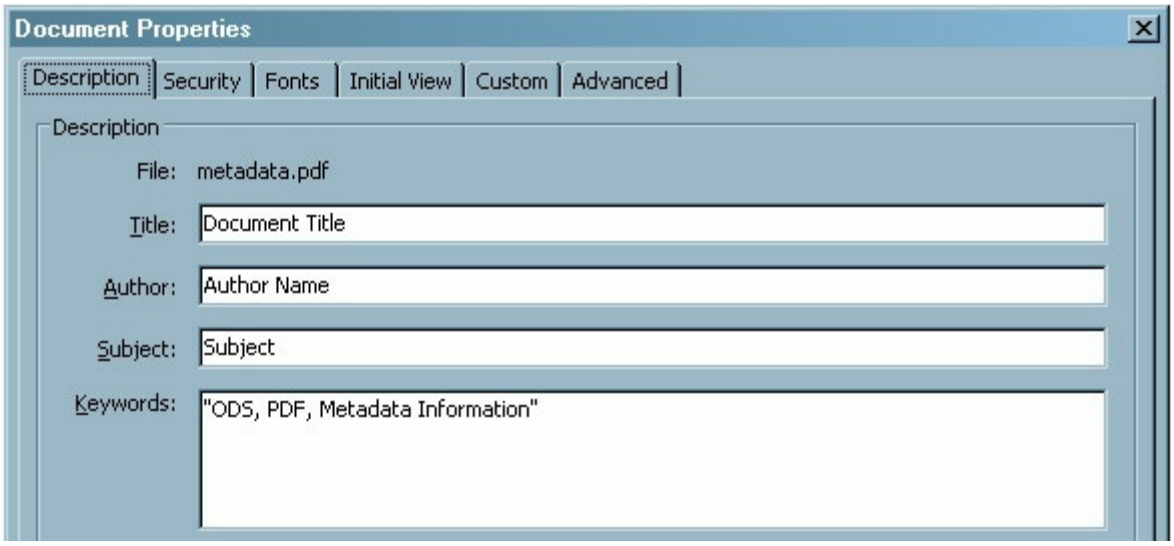
In the discussion of RTF and HTML output, we already saw some ODS option statements that inserted metadata information into the document properties area of the RTF output file or the <HEAD> area of the HTML output file. Similar options exist for PDF output as well.

The options that are available with ODS PDF are shown in the following code snippet:

```
ODS PDF FILE="metadata.pdf" AUTHOR="Author Name"
   SUBJECT="Subject" TITLE="Document Title"
   KEYWORDS="ODS, PDF, Metadata Information";
  *. . .SAS Code to test . . .;
ODS PDF CLOSE;
```

The procedure that we run is irrelevant, as the metadata information is only visible in the document properties of the PDF file, as shown in Figure 5.14. To review the document properties of your PDF file, select **File u Document Properties** from the pull-down menu (or CTRL + D in Adobe Reader 7 and later).

Figure 5.14



These options are designed to work primarily with the ODS PDF destination. They also work when you create a PostScript file using ODS PS or ODS PRINTER, but you must also use the PDFMARK option so that the correct document information is created when you distill the file into a PDF file format.

Creating a Linked Set of PDF Results

By default, the ODS PDF destination creates a set of bookmarks in your PDF output, and the output opens in Adobe Reader with a table of contents on the left that links to these bookmarks, as shown in Figure 5.15. We've also seen, earlier in this chapter, how ODS PDF will create a separate "Table of Contents" page. However, sometimes you want to generate your own, perhaps more abbreviated, contents information.

An alternative approach is to use the NOTOC option in the ODS PDF statement to turn this default bookmarking and table of contents off and instead create your own table of contents. The following is a simple example. There are two sections of code. The first starts the PDF file and sets up a page for the table of contents.

```
      ODS ESCAPECHAR="^";
ODS PDF FILE="d:\temp\book\Reports.pdf" NOTOC;
title "Reports";
ODS PDF TEXT="^S={JUST=C URL='#sales'}Sales";
ODS PDF TEXT="^S={JUST=C URL='#revenue'}Revenue";
ODS PDF STARTPAGE=NOW;
```

The preceding code uses a TITLE statement and two ODS TEXT= statements to create a first page in the report with a title and two hyperlinks that have meaningful labels. The STARTPAGE= option creates a page break so the report output starts on the following page. The second section of code, which follows, builds the rest of the report.

```
      title "Sales Report";
ODS PDF ANCHOR="sales";
proc means data=gallery sum;
   class gallery;
   var sales;
run;
title "Revenue Report";
ODS PDF ANCHOR="revenue";
proc means data=gallery sum;
   class gallery;
   var revenue;
run;
ODS PDF CLOSE;
```

The ANCHOR= option before each procedure marks a place in the output for the hyperlinks and must be the same as the URL= value specified with the escape sequence. When users click on a link in the title page, as shown in the boxed area in Figure 5.16, they are linked to the correct report in the following pages, as shown in Figure 5.17.

This is a very simple example. This technique can be enhanced with additional formatting added to the ODS TEXT= statements, specification of more descriptive labels, and additional anchors. With a little more work, you can create a highly formatted and attractive table of contents to guide readers through a complex report.

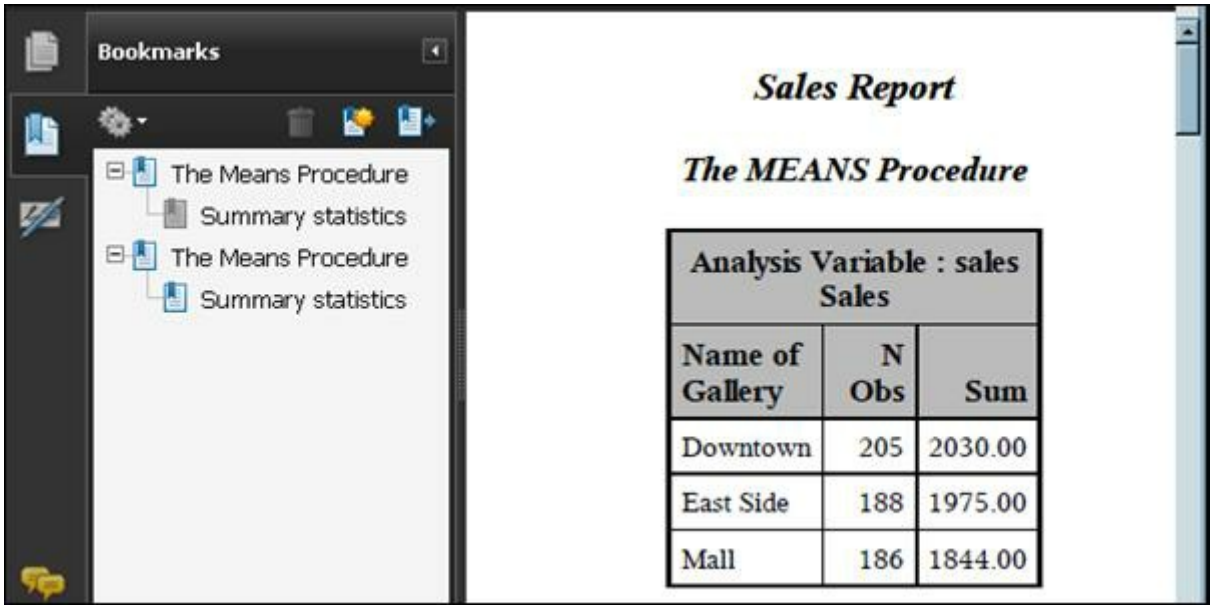


Figure 5.15

Figure 5.16: Only Links Shown from Page

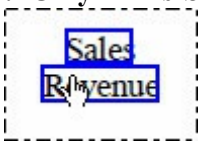
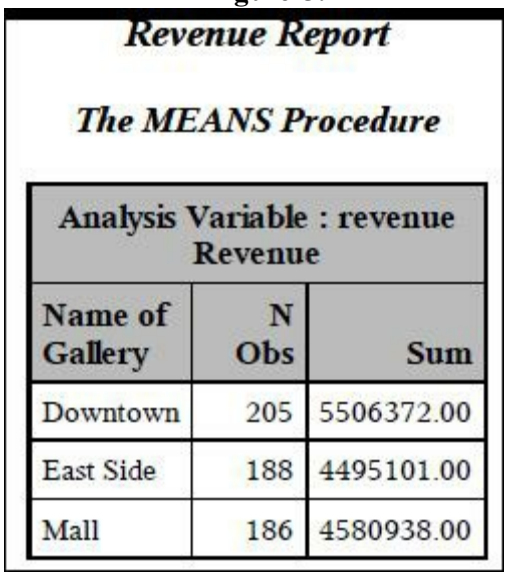


Figure 5.17



Revenue Report

The MEANS Procedure

Analysis Variable : revenue Revenue		
Name of Gallery	N Obs	Sum
Downtown	205	5506372.00
East Side	188	4495101.00
Mall	186	4580938.00

Adding “Page X of Y” Numbers to PDF Output

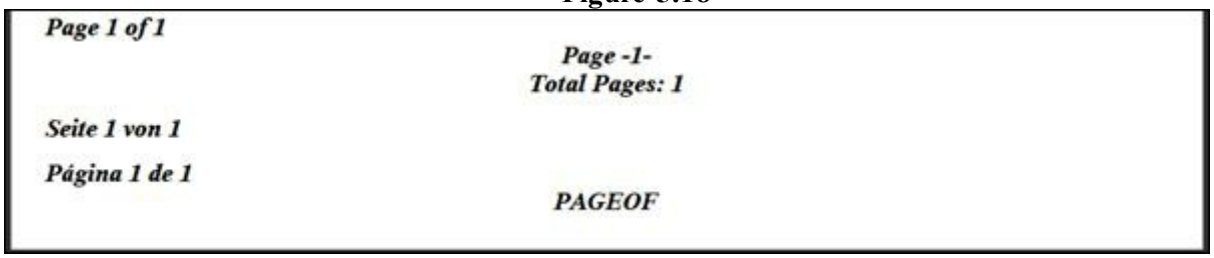
In an ODS RTF example, we defined an ODS escape character (ODS ESCAPECHAR=) so that we could code a special sequence to insert page numbers into the document in the form “Page X of Y”. This capability is also available when creating PDF output with ODS. The ESCAPECHAR functions to use with ODS PDF output are {thispage} for the “X” position and {lastpage} for the “Y” position. Each ESCAPECHAR function is enclosed in curly braces and preceded by the escape character. In the following code, the ODS escape character is set to the caret (^).

Consider the following set of FOOTNOTE statements:

```
ODS ESCAPECHAR='^';
ODS PDF FILE="Footnotes.pdf";
footnote justify=left 'Page ^{thispage} of ^{lastpage}';
footnote2 'Page -^{thispage}-';
footnote3 'Total Pages: ^{lastpage}';
footnote5 justify=left 'Seite ^{thispage} von ^{lastpage}';
footnote7 justify=left 'Página ^{thispage} de ^{lastpage}';
footnote8 '^{pageof}';
proc print data=gallery(obs=20);
run;
ODS PDF CLOSE;
```

They produce the output shown in Figure 5.18. Note that the {pageof} function when used with PDF does NOT produce the same results as when used with the ODS RTF destination. The {pageof} escape character function is available only when sending output to the RTF destination.

Figure 5.18



Controlling Page Breaks

As with RTF output, you can turn page breaks on and off for the printer destinations. This allows you to save some space if you have a procedure that doesn't generate much output. By turning off page breaks, you can start the next procedure's output on the same page.

To turn page breaks off, use the STARTPAGE=NO option. For example, if you have a small table produced by the MEANS procedure, and you'd like to continue your report with the results from a PROC UNIVARIATE step on the same page, the following code will do the trick:

```
      ODS PDF FILE='startpage.pdf' STARTPAGE=NO;
proc means data=gallery maxdec=0;
   var sales revenue;
run;
proc univariate data=gallery normal;
   var sales revenue;
run;
ODS PDF CLOSE;
```

This code produces the first page of output, part of which is shown in Figure 5.19. The output from the MEANS procedure is shown at the top, and then it continues with the first three tables from the first PROC UNIVARIATE step. The remaining tables produced by the PROC UNIVARIATE step continue over subsequent pages.

This option setting allows you to maximize the use of every page of output. ODS starts a new page only when it runs out of space on the current page. However, sometimes you want to force a page break. For example, suppose you want to force a break between the PROC UNIVARIATE analyses of SALES and REVENUE. To do this, separate the code for the two variables into two PROC UNIVARIATE steps, and insert the ODS PDF STARTPAGE=NOW statement between the two steps. Now the second set of PROC UNIVARIATE results starts on a new page, as shown in
Figure 5.20.

```
      ODS PDF FILE='startpage2.pdf' STARTPAGE=NO;
proc means data=gallery maxdec=0;
   var sales revenue;
run;
proc univariate data=gallery normal;
   var sales;
run;
ODS PDF STARTPAGE=NOW;
proc univariate data=gallery normal;
   var revenue;
run;
ODS PDF CLOSE;
```

The STARTPAGE=NOW option inserts a single page break, and the STARTPAGE=NO option remains in effect. Subsequent procedures will only start a new page when there is no space remaining on the current page unless you specify another ODS PDF STARTPAGE=NOW statement.

Figure 5.19

The MEANS Procedure

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
sales	Sales	579	10	6	2	23
revenue	Revenue	579	25186	20824	1040	112608

The UNIVARIATE Procedure
Variable: sales (Sales)

Moments			
N	579	Sum Weights	579
Mean	10.1018998	Sum Observations	5849
Std Deviation	5.58714973	Variance	31.2162421
Skewness	0.78319869	Kurtosis	-0.2761975
Uncorrected SS	77129	Corrected SS	18042.9879
Coeff Variation	55.3079106	Std Error Mean	0.23219402

Figure 5.20

Extreme Observations			
Lowest		Highest	
Value	Obs	Value	Obs
2	501	23	373
2	416	23	426
2	394	23	508
2	275	23	549
2	227	23	572

The UNIVARIATE Procedure
Variable: revenue (Revenue)

Variable: revenue (Revenue)

Moments			
N	579	Sum Weights	579
Mean	25185.5112	Sum Observations	14582411
Std Deviation	20823.7617	Variance	433629051
Skewness	1.30095323	Kurtosis	1.71743979
Uncorrected SS	6.17903E11	Corrected SS	2.50638E11
Coeff Variation	82.6815128	Std Error Mean	865.406003

Sending ODS Output to Your Printer

If you want to send your output directly to your default printer from Windows, all you need is the following code:

```
      ODS PRINTER;
title 'Monthly Complaint Summary';
proc freq data=complaints;
  tables location*complaint / norow nocol nopct;
run;
ODS PRINTER CLOSE;
```

The first ODS statement starts routing output to your SAS default printer as defined by the SYSPRINT= SAS system option. The second ODS statement ends the file and submits the print job. If you forget the second ODS statement, your file will not print until you end your SAS session.

The standard listing output for this code is shown in Figure 5.21. It's a basic crosstabulation of two variables. The listing output is adequate, but it has some problems. For example, notice how the column labels for several of the categories are wrapped onto two lines in an unappealing fashion. Contrast the listing output with the Printer output shown in Figure 5.22. It has exactly the same information, but it's much more inviting. In addition, notice that the column widths have been adjusted so that none of the words wrap. ODS is quite an improvement.

In terms of readability, Printer output is probably your best option among the ODS destinations. Though the appearance of your output will vary somewhat depending on your printer, you can be assured that it will fit on the page (unlike RTF and HTML, which are prone to awkward page breaks).

However, there are some drawbacks. First, you can't edit your results. Unless you're creating PDF output (see the example later in this chapter), you can't change the printed page or the print file. Second, if you are not in a Windows environment, sending output directly to a printer isn't quite as easy as in this example. Information about other operating systems is covered briefly in this chapter, and in more detail in the Appendix, "Operating System Differences."

Actually, the statement that ODS PRINTER sends output to your default printer is not strictly true. ODS first checks your option settings. On Windows, ODS checks the SYSPRINT= option to see if a printer is specified. On other systems, ODS checks the PRINTERPATH= option. Only if these options are not set does SAS use your default printer.

If you use a network printer, you need to be careful about submitting ODS PRINTER commands like this. If you make a mistake in your code, you could send hundreds of pages to the printer, annoying other users. See the examples later in this chapter on creating an output file instead.

Figure 5.21

Monthly Complaint Summary

The FREQ Procedure

Table of Location by Complaint

Location(Store Location) Complaint(Complaint Category)

Frequency	Damaged	Not Delivered	Not Satisfied	Price	Salesperson	Wrong Product	Total
Boston	52	48	48	52	51	41	292
New York	28	35	43	38	25	27	196
Washington, D.C.	19	16	15	17	8	19	94
Total	99	99	106	107	84	87	582

Figure 5.22

Monthly Complaint Summary

The FREQ Procedure

Frequency

Table of Location by Complaint

Location(Store Location)	Complaint(Complaint Category)						
	Damaged	Not Delivered	Not Satisfied	Price	Salesperson	Wrong Product	Total
Boston	52	48	48	52	51	41	292
New York	28	35	43	38	25	27	196
Washington, D.C.	19	16	15	17	8	19	94
Total	99	99	106	107	84	87	582

Selecting a Printer

The previous example showed how to print output to your default printer. You could send your output to another printer by closing SAS, changing the default printer, and launching SAS again. However, there is an easier way to switch printers: the PRINTER= option.

This option is used in the first ODS PRINTER statement:

```
ODS PRINTER PRINTER=printername;
```

The value you fill in for *printername* is the name assigned to the printer that you want to use. This can be a local printer, a network printer, or it could even refer to a file on disk.

You can find out the names of all of your available printers on the Windows host by selecting **File** and then **Print** from within SAS. This opens up a print dialog box that has a list of all of the available printer names in a pull-down menu at the top. An example is shown in Figure 5.23. In this example, if you wanted to print to the highlighted printer, then the syntax would be as follows:

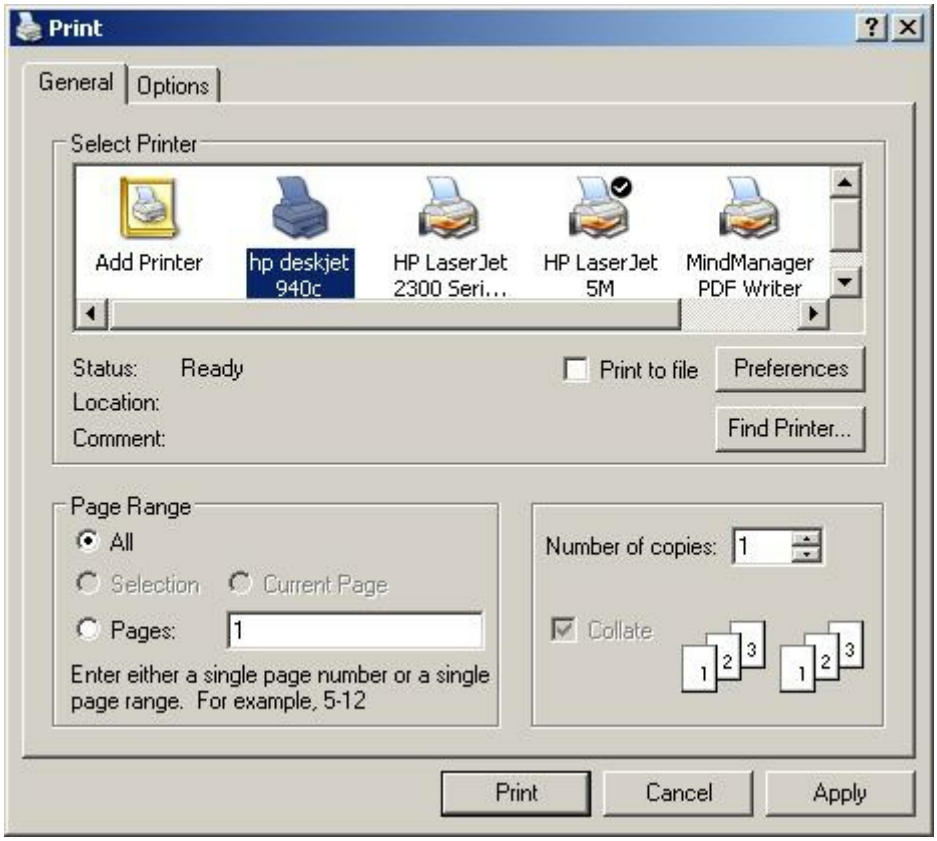
```
ODS PRINTER PRINTER="hp deskjet 940c";
```

You can take advantage of the PRINTER= option to send your Printer output to more than one printer in the same job. For example, if you have a summary report that you want to send to a high-quality color printer, and a long detail report you want to send to a line printer, you can use two PRINTER= statements.

```
    title 'Complaint Resolution Summary';
ODS PRINTER PRINTER="FancyColorPrinter;"
proc freq data=Complaints;
  tables Outcome / nocum;
run;
ODS PRINTER CLOSE;

title 'Complaint Resolution Detail';
ODS PRINTER PRINTER="LinePrinter";
proc print data=Complaints noobs label;
  var complaint outcome;
run;
ODS PRINTER CLOSE;
```

In this example, the output is sent to two printers. One is called “FancyColorPrinter” and the other is called “LinePrinter.” Once you learn the names of your printers, it’s easy to send your results exactly where you want them to print.

Figure 5.23


Print
General
Options
Select Printer
Add Printer
hp deskjet 940c
HP LaserJet 2300 Seri...
HP LaserJet 5M
MindManager PDF Writer
Status: Ready
Location:
Comment:
Print to file
Preferences
Find Printer...
Page Range
All
Selection
Current Page
Pages: 1
Enter either a single page number or a single page range. For example, 5-12
Number of copies: 1
Collate
Print
Cancel
Apply

Creating a PostScript File from Your ODS Printer Output

ODS PDF is not the only way to create a file that you can then print at your convenience. Another option is PostScript. The syntax for creating a PostScript file is to use one of these alternatives:

ODS PRINTER PS FILE=*fileref* or ODS PS FILE='*file-specification*'

Either method tells ODS to send your output to a specific file instead of the default printer. For the first example, PS is an option being sent to the PRINTER destination. In the second example, PS is the ODS destination. You could also have coded the second example as:

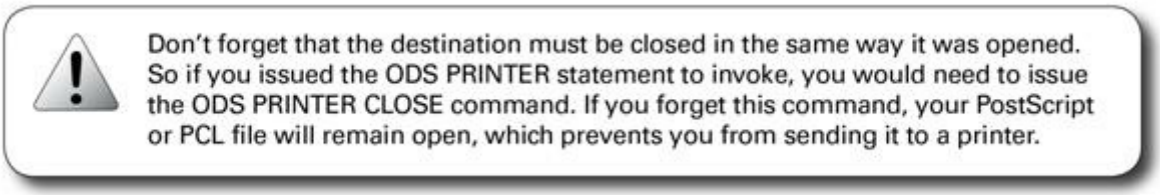
ODS POSTSCRIPT *<rest of statement>*

The following code shows how this works. It creates a PostScript file from a simple PROC PRINT step. This example is set up for the Windows environment. The file naming convention varies by operating system.

```
      ODS PRINTER PS FILE='DamagedCD.ps';
title 'Damaged CDs Report';
proc print data=complaints label noobs;
        where product='CDs';
  id location;
  var outcome;
run;
ODS PRINTER CLOSE;
```

If you print the file DamagedCD.ps, you get the output shown in Figure 5.24. The technique for printing this file varies by operating system. See the Appendix, "Operating System Differences," for information about your system.

This example covers how to create PostScript files. You also have the option of creating Printer Control Language (PCL) files, if that is what your printer requires. Unless you have a very old printer, it is unlikely that it will have any difficulty with PostScript files, but check your owner's manual if you have problems printing. To get a PCL file, use an ODS PCL statement instead of ODS PRINTER. ODS currently supports PCL1 through PCL5. See the online documentation for details on this option.



Don't forget that the destination must be closed in the same way it was opened. So if you issued the ODS PRINTER statement to invoke, you would need to issue the ODS PRINTER CLOSE command. If you forget this command, your PostScript or PCL file will remain open, which prevents you from sending it to a printer.

If you have problems printing the resulting PostScript file, see the example on the following page that discusses PostScript output creation.

Figure 5.24

Damaged CDs Report

Store Location	Outcome of Complaint
New York	Refund
Boston	Refund
Boston	Refund
Boston	Refund
New York	Unresolved
Boston	Refund
Boston	Refund
Boston	Unresolved
Boston	Unresolved
Boston	Apology
Boston	Refund
New York	Apology
New York	Apology
Boston	Refund
Washington, D.C.	Apology
Washington, D.C.	Refund

Using the SAS Printer Drivers

In Windows, when you print to your default printer, or specify a particular printer to use, ODS uses the driver for that printer to generate your output. This means that your output will vary slightly depending on which printer you select.

On all other operating systems, ODS uses printer drivers that are supplied by SAS to generate output. If you'd like your SAS output to be generated the same way on all operating systems, you can add the SAS option to your ODS PRINTER statement when running SAS on Windows. This statement has no effect on other operating systems.

```
ODS PRINTER SAS;
```

If you do not want to use the SAS printer drivers, you can choose the opposite option setting: HOST. The default option setting of SAS or HOST is set when you first install SAS, and the setting is based on your operating system. Unless you are having problems with Printer output, you probably do not want to change this setting. On Windows, it is safe to use either setting. On other systems, the HOST setting might cause ODS PRINTER to fail because no host drivers can be found.

Even if you do change the setting successfully, you are unlikely to see any noticeable difference. Figure 5.25 and Figure 5.26 show the results of the HOST and SAS options. Other than minor margin differences, they are identical.

Using the SAS PostScript Driver

If you're printing to PostScript printers, you have the option of using a generic PostScript driver supplied by SAS. This will create a standard PostScript file that you can print on any PostScript printer. This avoids the problem of creating a file based on your default printer at the time, and then having problems printing it to a different make or model of PostScript printer.

To request this driver, include the PRINTER=POSTSCRIPT option in your ODS PRINTER statement. Better yet, you can use the shorthand option PS, or just use ODS PS (dropping the PRINTER). The following three statements are equivalent:

```
ODS PRINTER SAS PRINTER=POSTSCRIPT;
ODS PRINTER PS;
ODS PS;
```

Like the SAS and HOST options, the PS option is not likely to make much difference. Figure 5.27 shows the effect of adding the PS option to the run that produced Figure 5.25. Again, there is no noticeable difference. Generally you would use this option only if you plan to print your file on multiple printers, or if you intend to view or distill your file using a third-party software product like Adobe Acrobat, Ghostview, or GSView.

```
ODS PRINTER PS FILE='MySharedFile.ps';
```

On the other hand, if you're printing directly to a single printer from Windows, you'll probably get the best results by using the driver for that specific printer. In this case, you should not use the SAS or PS option.

Figure 5.25: HOST Option

Delivery Failures

Store Location				
Location	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Boston	48	48.48	48	48.48
New York	35	35.35	83	83.84
Washington, D.C.	16	16.16	99	100.00

Figure 5.26: SAS Option

Delivery Failures

Store Location				
Location	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Boston	48	48.48	48	48.48
New York	35	35.35	83	83.84
Washington, D.C.	16	16.16	99	100.00

The PS option might be useful if you're having trouble getting your ODS printer output to print or display correctly. Sometimes your Windows printer driver might cause problems with ODS printer output. Using the generic SAS PostScript driver is one possible solution (checking online for an updated driver for your printer is another).

Figure 5.27: PS Option

Delivery Failures

Store Location				
Location	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Boston	48	48.48	48	48.48
New York	35	35.35	83	83.84
Washington, D.C.	16	16.16	99	100.00

Printing in Color

In the chapters on HTML and RTF output, you have seen that ODS generates attractive color output. You can do the same with ODS PRINTER output. However, if you're printing to a black and white laser printer, this isn't very useful.

You use two option settings to create ODS PRINTER output that is tailored to your printer and to the style of output you would like. You can have simple black and white "book style" output with black text, black table borders, and no shading. Alternatively, you can have output that uses color for the text, table borders, and shading. Yet another choice is to create gray-scale output that looks somewhat like the full-color output, but will print on a black and white printer.

By default, SAS tries to generate color output. An example of this is shown in Figure 5.28. If the printer you select is not a color printer, then gray-scale output that imitates color is created. An example of this is shown in Figure 5.29.

The reason you get this gray-scale output is the default setting of two options. First, the system option COLORPRINTING is turned on by default. Second, the COLOR= option in the ODS PRINTER statement defaults to YES. If you don't like the output shown in Figure 5.28 and Figure 5.29, then you need to change one or both of these options. If you just want to turn off color printing for a particular set of results, you can use the COLOR=NO option in the ODS PRINTER statement.

```
ODS PRINTER COLOR=NO STYLE=DEFAULT;
```

Alternatively, the option can be specified as follows:

```
ODS PRINTER NOCOLOR STYLE=DEFAULT;
```

The STYLE= option is used here to specify a style definition that uses color to better illustrate this example. This creates the printed output shown in Figure 5.30.

If you never want color output, you should turn off the COLORPRINTING SAS system option by issuing the following command. This way you don't have to remember to add the NOCOLOR option every time you use ODS PRINTER.

```
OPTIONS NOCOLORPRINTING;
```

With this option in place, your output looks like Figure 5.30 if you have specified COLOR=NO, and it looks like Figure

Figure 5.28: COLOR=YES Sent to Color Printer

Delivery Failures

Store Location				
Location	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Boston	48	48.48	48	48.48
New York	35	35.35	83	83.84
Washington, D.C.	16	16.16	99	100.00

Figure 5.29: COLOR=YES Sent to Black and White Printer

Delivery Failures

Store Location				
Location	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Boston	48	48.48	48	48.48
New York	35	35.35	83	83.84
Washington, D.C.	16	16.16	99	100.00

Figure 5.30: COLOR=NO (Regardless of Printer Type)

Delivery Failures

Store Location				
Location	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Boston	48	48.48	48	48.48
New York	35	35.35	83	83.84
Washington, D.C.	16	16.16	99	100.00

Creating Consistent Printed Pages

If your output runs to more than one page, you might want to think about using the ODS UNIFORM option. This option calls for ODS to use the same column widths across all pages of your output. If you're creating a long report, this option creates a more consistent look for all of the pages.

However, it comes with a price. In order to figure out the best column width to use, the ODS UNIFORM option forces ODS to read the entire table before creating the first page of output. This requires additional memory. If you have an extremely long set of results or have limited memory available, you might want to skip this option. To see how this option makes a difference, look at the following example:

```
      ODS PDF FILE='No_Uniform.pdf';
ODS PRINTER;
proc print data=Uniforms noobs label;
  var UniformType Color Sales;
run;
ODS _ALL_ CLOSE;
```

Part of the output for this code is shown in Figure 5.31. It shows the tops of the pages 1 and 7. Notice how the column heading for Type of Uniform on page 7 is much wider than the column heading on page 1. If this doesn't bother you, then don't worry about the UNIFORM option. However, if you would like consistent output, turn on the UNIFORM option.

The syntax for the UNIFORM option is simple; just add the option to your ODS PDF or ODS PRINTER statement, as follows:

```
      ODS PDF FILE='Uniform.pdf';
ODS PRINTER UNIFORM;
proc print data=Uniforms noobs label;
  var UniformType Color Sales;
run;
ODS _ALL_ CLOSE;
```

The new printout is shown in Figure 5.32. The only change is that now the column heading is wide on both page 1 and page 7. Because the UNIFORM option was turned on, ODS reviewed the entire table before picking a column width. Because page 7 required a wider column heading, ODS applied that wider column heading to page 1 as well.

If you have an extremely long table, but still want the column widths to be consistent, you could look into editing the table definition to force the width you want. This is an advanced technique that is covered in the section on style templates.

This example points out another thing to keep in mind about the UNIFORM option. In addition to using more memory, it also uses more space on the printed page. ODS applies this option by using the widest column size across all pages. This might mean that your output requires

extra pages. The UNIFORM option works with all the destinations in the ODS PRINTER family of destinations, including ODS PDF.

Figure 5.31

Page 1

Type of Uniform	Color of Uniform	Uniforms Sold
Maid	White	181
Maid	Red	372
Maid	Red	415
Maid	White	138
Maid	Green	91
Maid	Red	364

Page 7

Type of Uniform	Color of Uniform	Uniforms Sold
Extremely Pretentious Doorman	White	115
Extremely Pretentious Doorman	White	272
Extremely Pretentious Doorman	Red	347
Extremely Pretentious Doorman	White	236
Extremely Pretentious Doorman	Red	313
Extremely Pretentious Doorman	White	225

Figure 5.32

Page 1

Type of Uniform	Color of Uniform	Uniforms Sold
Maid	White	181
Maid	Red	372
Maid	Red	415
Maid	White	138
Maid	Green	91
Maid	Red	364

Page 7

Type of Uniform	Color of Uniform	Uniforms Sold
Extremely Pretentious Doorman	White	115
Extremely Pretentious Doorman	White	272
Extremely Pretentious Doorman	Red	347
Extremely Pretentious Doorman	White	236
Extremely Pretentious Doorman	Red	313
Extremely Pretentious Doorman	White	225

Creating Multiple Columns on a Page

Just as you can for the RTF destination, it is possible to create multi-column reports for the PRINTER destinations. When you realize that you have a tall skinny table, one thing you can do is display it in multiple columns so that you can get more data on a given page. The syntax is simple; just add the COLUMNS= option. Note that the COLUMNS= option works for all the ODS PRINTER family destinations, including ODS PDF.

```
      ODS PDF FILE='Use_Columns.pdf' COLUMNS=3;
ODS PRINTER COLUMNS=3;
proc print data=gallery label;
        var gallery artist price;
      run;
ODS _ALL_ CLOSE;
```

By setting COLUMNS=3, we get the table shown in Figure 5.33. Now the page is being used effectively to fit the output. Also, notice that the table headers repeat at the top of the new column.

However, too many columns will just make the page hard to read. Another limitation to keep in mind when using the COLUMNS= option is that it's up to you to determine whether your output will fit into the multiple columns. If your table has too many variables, SAS will fit it into the number of columns you specify, and the results might not be pretty.

Controlling Page Orientation

When you have a table that is tall and skinny, it makes sense to print in portrait layout. But when you have a table that is wider than it is tall, a landscape layout makes more sense. By default, if you don't change any options, the ODS PRINTER destinations create portrait output. Just as with RTF output, you can change the orientation by using the system option ORIENTATION=.

```
      OPTIONS ORIENTATION=LANDSCAPE;
```

However, it is not possible to switch orientations within a single instance of ODS Printer or ODS PDF output. Although this capability is supported for RTF output, a single Printer family file created with ODS must be the same orientation throughout. A single PDF file created with Landscape orientation is shown in Figure 5.34. Here is the program that produced the output:

```
      OPTIONS ORIENTATION=LANDSCAPE;
      ODS PDF FILE='Chg_Orientation.pdf';
ODS PRINTER;
      proc print data=gallery label;
        var gallery artist price;
      run;
      proc means data=gallery;
      run;
      ODS _ALL_ CLOSE;
```

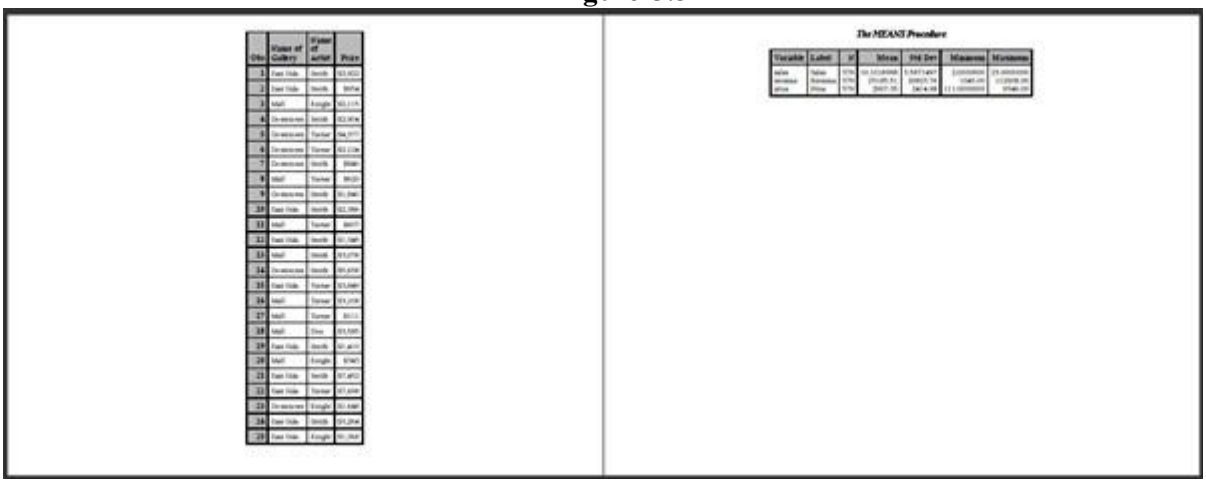
Figure 5.33

Obs	Name of Gallery	Name of Artist	Price
1	East Side	Smith	$2,922
2	East Side	Smith	$954
3	Mall	Knight	$2,115
4	Downtown	Smith	$2,904
5	Downtown	Turner	$4,377
6	Downtown	Turner	$2,124

Obs	Name of Gallery	Name of Artist	Price
35	East Side	Doe	$896
36	East Side	Doe	$3,111
37	Mall	Doe	$2,334
38	Mall	Smith	$1,248
39	Downtown	Turner	$3,588
40	Mall	Turner	$477

Obs	Name of Gallery	Name of Artist	Price
69	East Side	Smith	$284
70	East Side	Turner	$2,300
71	Mall	Smith	$506
72	Mall	Doe	$2,184
73	Mall	Smith	$1,164
74	East Side	Doe	$808

Figure 5.34



Controlling Page Margins

Setting custom page margins is even easier in ODS PRINTER output than it was for RTF output. All you have to do is adjust the settings for the SAS system options for margins: TOPMARGIN=, BOTTOMMARGIN=, LEFTMARGIN=, and RIGHTMARGIN=. For example, for smaller margins, use the following:

```
OPTIONS LEFTMARGIN=.25in RIGHTMARGIN=.25in
 TOPMARGIN=.25in BOTTOMMARGIN=.25in;
```

With the preceding options and OPTION NOCENTER also in effect, a PROC PRINT of data set GALLERY looks as shown in Figure 5.35. Notice that the top and left margins are now 0.25 inches. The right margin looks wider, but that is because the table is not wide enough to use the entire page width. These settings enable you to maximize the available space on a page. To make the margins look more uniform, specify the CENTER system option to center the table on the page, which creates equal right and left margins.

Notice that unlike the RTF destination, there are no margin settings built into the default style for Printer output, so you do not need to modify the style for these margin settings to work.

Controlling Page Size

Although you can't use PAGESIZE to control page size in ODS PRINTER output, you do have some control over this setting. Just as in an RTF example in the previous chapter, you can use the PAPERSIZE= option to choose a different page size for your output.

For example, if your output is so wide that even using ORIENTATION=LANDSCAPE is not enough help to get your results to fit on the page, one option is to change to wider paper. You can switch to a page size appropriate for legal-size paper. When you print your document, it will be set up for legal-size paper. The syntax for requesting a different paper size is simple:

```
OPTIONS PAPERSIZE=LEGAL;
```

To use legal-size paper to fit a wide table onto the page, try the following syntax. It sets the orientation to Landscape and the paper size to Legal. The results are shown in Figure 5.36.

```
    OPTIONS ORIENTATION=LANDSCAPE PAPERSIZE=LEGAL;
ODS PDF FILE='Use_Papersize.pdf';
ODS PRINTER;
proc means data=clips maxdec=1
  mean std min p1 p5 p10 p25 p50 p75 p90 p95 p99 max;
  class tone;
  var mentions compete;
run;
ODS _ALL_ CLOSE;
```

To set the PAPERSIZE= option back to letter-size paper, submit the following OPTIONS statement:

```
OPTIONS PAPERSIZE=LETTER;
```

Figure 5.35

Obs	Limited Edition	Medium	Name of Artist	Name of Gallery	Sales	Revenue	Price
1	Yes	Oil	Smith	East Side	3	$8,766	$2,922
2	Yes	Oil	Smith	East Side	7	$6,678	$954
3	Yes	Watercolor	Knight	Mall	6	$12,690	$2,115
4	Yes	Oil	Smith	Downtown	6	$17,424	$2,904
5	Yes	Watercolor	Turner	Downtown	4	$17,508	$4,377

Figure 5.36

The MEANS Procedure

Tone of content	N Obs	Variable	Label	Mean	Std Dev	Minimum	1st Pctl	5th Pctl	10th Pctl	25th Pctl	50th Pctl	75th Pctl	90th Pctl	95th Pctl	99th Pctl	Maximum
Negative	105	Mentions	Number of references to company	5.8	3.5	1.0	1.0	1.0	1.0	3.0	5.0	8.0	11.0	12.0	12.0	12.0
		Compete	Number of references to competition	3.0	1.7	1.0	1.0	1.0	1.0	2.0	4.0	5.0	6.0	6.0	6.0	6.0
Neutral	111	Mentions	Number of references to company	6.2	3.7	1.0	1.0	1.0	1.0	3.0	6.0	10.0	11.0	12.0	12.0	12.0
		Compete	Number of references to competition	3.2	1.7	1.0	1.0	1.0	1.0	2.0	3.0	5.0	6.0	6.0	6.0	6.0
Positive	114	Mentions	Number of references to company	7.0	3.0	1.0	1.0	2.0	3.0	5.0	7.0	9.0	11.0	12.0	12.0	12.0
		Compete	Number of references to competition	3.2	1.7	1.0	1.0	1.0	1.0	2.0	3.0	4.0	6.0	6.0	6.0	6.0

Creating Multiple Versions of the Same Output

One of the best features of the ODS PRINTER destinations is that you can print your output directly, or save it to a file. You can also do both in the same step: you can immediately generate a hard copy for distribution and also create a file as a permanent record.

For example, the first PROC MEANS step sends output directly to the printer. The second PROC MEANS step, which is identical to the first, creates a PostScript file of the same results.

```
      ODS PRINTER;
proc means data=gallery maxdec=1 mean min max;
  class medium;
  var price;
run;
ODS PRINTER CLOSE;

      ODS PRINTER PS FILE='FileCopy.ps';
proc means data=gallery maxdec=1 mean min max;
  class medium;
  var price;
run;
ODS PRINTER CLOSE;
```

This gives you the printed output shown in Figure 5.37 and the PostScript file shown in Figure 5.38. The PostScript file can be viewed or printed at a later date and will look the same as the original printed output.

However, to get this result, we had to run the procedure twice. Not a big deal if the data set is small and the procedure is simple. However, it wastes resources. It could also introduce an error if you do not ensure that the two steps are exactly the same.

A better way to create multiple versions of the same output is to run them both at the same time. This can be done by supplying an ID number for each run.

```
      ODS PRINTER(1);
ODS PRINTER(2) PS FILE='GalleryCopy.ps';
proc means data=gallery maxdec=1 mean min max;
  class medium;
  var price;
run;
ODS _ALL_ CLOSE;
```

The ODS PRINTER(1) command opens an output destination to print hard copy directly to the default printer. The ODS PRINTER(2) command opens another destination to create the PostScript file. You can do them both at the same time. To close both destinations, you could type ODS PRINTER CLOSE statements for each destination, or use the _ALL_ option to close them both at once. The ODS _ALL_ CLOSE statement also closes any other open destinations.

The syntax ODS PRINTER(1) is shorthand for ODS PRINTER(ID=1). You can also

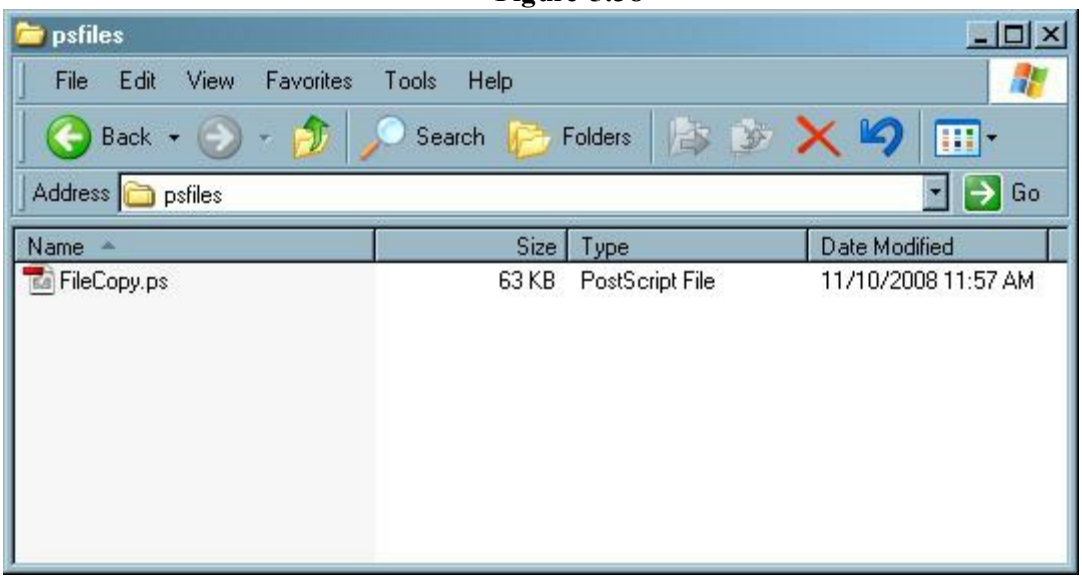
specify more descriptive identifiers. For example, you could specify ODS PRINTER(ID=HardCopy) and ODS PRINTER(ID=PostScript). As long as the identifiers are unique, you will get your different versions.

Figure 5.37

The MEANS Procedure

Analysis Variable : price Price				
Medium	N Obs	Mean	Minimum	Maximum
Oil	393	3489.4	202.0	9546.0
Watercolor	186	1677.6	111.0	4617.0

Figure 5.38



In addition to using the two ODS PRINTER statements to create two different types of printer output, you can also use them to create two of the same type of printer output, but with different SELECT or EXCLUDE options in place. This would allow printing a short summary report while saving in a file the full report with all of the details in a single procedure call. All you need to do is use the same identifier in the ODS SELECT statement:

```
ODS PRINTER(1) SELECT statistic-name(s);
```

See Chapter 8, "Limiting ODS Output," for more information about ODS SELECT and EXCLUDE.

SAS System Options That Affect ODS Printer Output

This chapter has already examined the impact of two of the system options that affect ODS Printer output, the COLORPRINTING option and the ORIENTATION option. However, there are other system options that you should investigate. Most system options have some impact on your output.

Be aware that many of the options are dependent on your printer and operating system, and they all might not work for you. Some options are specific to the ODS PRINTER or ODS PDF destinations. To see the entire list of these options, submit the following program:

```
proc options group=(odsprint pdf);
run;
```

The results from the preceding program appear in the SAS log, as shown in Figure 5.39.

On Windows, the default settings for most of these options are based on the current settings of your default printer. So if you adjust the settings on your default printer, you should not need to alter these options. However, if you want these settings to vary within a single SAS session, you'll need to use the SAS system options.

Figure 5.39

```
    SAS (r) Proprietary Software Release 9.2  TS1M0

Group=ODSPRINT
 BINDING=DEFAULT   Controls the binding edge for duplexed output
 BOTTOMMARGIN=0.000 IN
                   Bottom margin for printed output
 NOCOLLATE         Do not collate multiple copies of printed output
 COLORPRINTING     Print in color if printer supports color
 COPIES=1          Number of copies to make when printing
 DEFLATION=6       Identify the level of deflate compression
 NODUPLEX          Do not print output in duplex mode
 FONTEMBEDDING     Enable font embedding by Universal Printing
 FONTRENDERING=FREETYPE_POINTS
                   Identify font rendering
 GSTYLE            Use ODS styles in the generation of graphs. GSTYLE does not affect ODS
                   styles for graphs generated with Java, ActiveX drivers or Statistical
                   Graphics
 JPEGQUALITY=75    Identify the level of JPEG compression
 LEFTMARGIN=0.000 IN
                   Left margin for printed output
 ORIENTATION=PORTRAIT
                   Orientation to use when printing
 PAPERDEST=        Bin to receive printed output
 PAPERSIZE=LETTER  Size of paper to print on
 PAPERSOURCE=      Bin to pull paper from when printing
 PAPERTYPE=PLAIN   Type of paper printer is using
 PRINTERPATH=      Printer path
 RIGHTMARGIN=0.000 IN
                   Right margin for printed output
 TEXTURELOC=!sasroot\common\textures
                   Location of textures and images used by ODS styles.
 TOPMARGIN=0.000 IN
                   Top margin for printed output
 NOUNIVERSALPRINT  Do not use Universal Printing Model
 UPRINTCOMPRESSION Enable Universal Printer compression
 NOPRTPERSISTDEFAULT
                   Disable persist of the printer selected in Print Setup between SAS sessions.
 SYSPRINT=("HP LaserJet 2300 Series PCL 5")
                   Set the default printer and, optionally, an alternate destination (file) for
                   output
 NOUPRINTMENUSWITCH
                   Disable universal print menus when UNIVERSALPRINT is on.
Group=PDF
 PDFACCESS         Allow access to PDF documents
 NOPDFASSEMBLY     Do not allow PDF document assembly
 NOPDFCOMMENT      Do not allow modification of PDF document comments
 NOPDFCONTENT      Do not allow modification of PDF document content
 PDFCOPY           Allow copying of PDF document text and graphics
 PDFFILLIN         Allow fields in a PDF document to be filled
 PDFPAGELAYOUT=DEFAULT
                   Identify the page layout for PDF documents
 PDFPAGEVIEW=DEFAULT
                   Identify page viewing mode for PDF documents
 PDFPASSWORD=xxxxxxxx
                   PDF open and owner passwords
 PDFPRINT=HRES     Identify PDF document printing permissions
 PDFSECURITY=NONE  Identify the level of PDF document encryption
```

Converting to PDF with Acrobat

Generally, using the ODS PDF statement produces perfect PDF output from your SAS results. However, you don't have much control over the conversion process. There are a number of decisions that go into creating a PDF file. You can let SAS make them for you, or if you want to invest in another piece of software, you can take control of the conversion yourself.

To create PDF files outside of SAS, you need to purchase software that distills PostScript files. Distilling is the term used to describe converting a file from PostScript to PDF. The most common distiller on the market is Adobe Acrobat. Unlike the free Adobe Reader, Adobe Acrobat is a commercial product that you must purchase.

If you install Adobe Acrobat, you will find that you have two new printer definitions available on your system: Acrobat PDFWriter and Acrobat Distiller. These can each be configured with various options. To change the default options on Acrobat PDFWriter, select **Printers** from the Control Panel and then modify the properties of Acrobat PDFWriter.

However, if you want full control over the conversion process, it is recommended that you use Acrobat Distiller. To change the default options for Acrobat Distiller, open the Acrobat Distiller software and select one of the three optimization settings (**Press**, **Print**, or **Screen**). Figure 5.40 shows the main Distiller window.

Next, select **Edit Adobe PDF Settings** from the **Settings** menu. This gives you access to dozens of conversion options. These options affect things such as how graphic images are processed and compressed, which is useful if you don't like the looks of your graphics output, and whether the PDF file is optimized for printing or screen display. Figure 5.41 shows the **Edit Adobe PDF Settings** window for Standard settings. These figures were made using Acrobat Distiller 9.

To send ODS output to Acrobat PDFWriter or Acrobat Distiller, use ODS PRINTER and the PRINTER= option. Do not use ODS PDF, as this will invoke the built-in PDF driver from ODS instead of using the Acrobat driver that you just customized.

```
ODS PRINTER PRINTER="Acrobat PDFWriter";
ODS PRINTER PRINTER="Acrobat Distiller";
```

The preceding code works if you are working on the system where Acrobat is installed. However, if Acrobat is not available for the operating system you are running, then creating customized PDF output is a two-step process.

First, you need to create a PostScript file from ODS. Use the ODS PS statement and specify an output file with the FILE= option.

```
ODS PS FILE="MyOutput.ps";
```

Then take that file to a computer with Adobe Acrobat installed and distill it there. On a Windows system, if you open a file with the .ps extension, Acrobat Distiller automatically launches and creates the PDF for you.

Figure 5.40

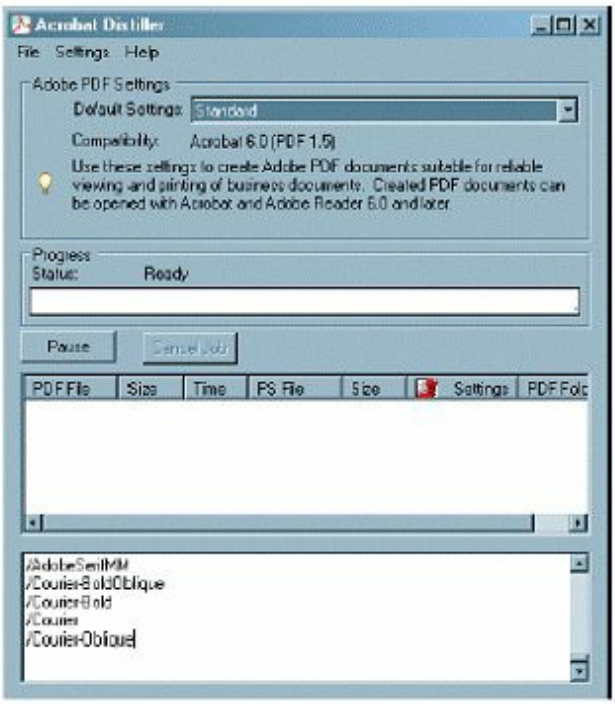
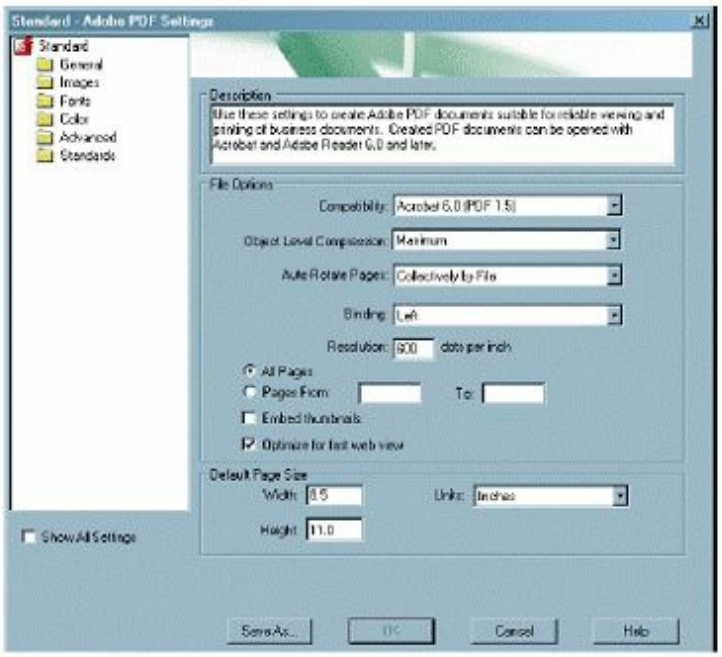


Figure 5.41



Adobe Acrobat is not the only product that can be used to create PDF files. There are several other freeware, shareware, and commercial products (GhostScript, for one) that also produce PDF output. Acrobat is the most widely used product.

If you want to include bookmarks in a PDF file you create by printing to PDFWriter or Distiller, or by distilling a PostScript file, you need to add the PDFMARK option to your ODS PRINTER statement.

Part 3
Output Structure

Chapter 6 Exploring ODS Output 197
Chapter 7 Output Data Sets 211
Chapter 8 Limiting ODS Output 235
Chapter 9 Using ODS in the DATA Step 255
Chapter 10 Creating Custom Reports from the DATA Step 271
Chapter 11 Modifying and Replaying ODS Output 293
Chapter 12 Using Tagsets to Generate MARKUP Output 321

Chapter 6
Exploring ODS Output

The previous chapters have displayed the output generated by a variety of ODS statements and options. The results you see in your Web browser, in a word processor, on your printer, or in the Results window are built from one or more output objects, which are the basic building blocks produced by SAS and used to create ODS report output.

This chapter explores these output objects in detail. Then we explore a number of procedure results to see how output objects are combined to produce ODS output.

This chapter provides important background information that you might need in order to understand the chapters to come.

For example, knowing how to identify output objects becomes helpful when you get to the next chapter, which shows how to select and exclude output objects from your result.

This chapter uses the data set SALES to illustrate the concepts. This data set is available on the author pages for this book at http://support.sas.com/authors.

The ODS TRACE Statement

A single SAS procedure can produce a single ODS output object, or it can produce dozens. In order to be able to customize your results, you need to be able to identify these output objects. The output objects from your procedure are what you see in the SAS Results window. ODS provides a statement that generates a description of each output object: the TRACE statement. Here is the syntax for this statement:

```
    ODS TRACE ON;
  *procedures and/or DATA steps go here;
ODS TRACE OFF;
```

This TRACE statement is used twice, the first time to turn output tracing on, and the second time to turn it off again. What happens in between is that ODS sends information about each output object to the SAS log. To see this statement in action, look at the following code:

```
    ODS TRACE ON;
proc freq data=sales;
  tables Department / chisq;
run;
ODS TRACE OFF;
```

Here we've asked for a single procedure with a single variable. We haven't specified any ODS destinations, so we get the default listing output shown in Figure 6.1. It looks like one piece of output. However, if you look at the SAS log, shown in Figure 6.2, you can see that there are actually two output objects. Each one is described in a section of the log titled "Output Added:". Figure 6.2 presents the SAS Results window next to the output from the SAS log. This shows how the output from the ODS TRACE statement reflects the same order as that of the icons in the Results window.

The first output object is the frequency table itself. The ODS TRACE information in the SAS log tells us that this part of the output is named "OneWayFreqs." It has a label of "One-Way Frequencies." You can think of the name and label just like variable names and labels. The only difference is that in this case the name and label refer to an output object. This information is important, because if you want to select or exclude part of your ODS output, you have to know how to identify it.

The ODS TRACE information also supplies the name of the table definition (template) that has been applied to this output object. The formatting of the frequency table is controlled by the table definition called "Base.Freq.OneWayFreqs." Notice how this table definition is named. It starts with "Base," because this is a Base SAS procedure. The next segment of the name is the procedure name: "Freq." This is followed by the name of the output object: "OneWayFreqs."

The final piece of information is the path. This is a specific roadmap to this output object. Every one-way frequency table will have the same name, label, and table definition. The path tells you how to identify this individual one-way frequency table. Notice how the pathname includes the table name. If the table statement had multiple variables listed, the SAS log

would reflect the same output object name for each variable, but each variable would have a different path. (In earlier versions of SAS, you might see the variable name instead of the table name. It's a good idea to always check the documentation to see whether output object names have changed between releases of SAS.)

The second set of TRACE information in Figure 6.2 shows the name, label, template, and path for the chi-square statistics. Notice how the naming conventions follow a similar pattern.

Figure 6.1

Department

Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Wholesale	134	26.80	134	26.80
Internet	208	41.60	342	68.40
Mail Order	87	17.40	429	85.80
Retail	71	14.20	500	100.00

Chi-Square Test for Equal Proportions

Chi-Square	90.6400
DF	3
Pr > ChiSq	<.0001

Sample Size = 500

Figure 6.2

```
Output Added:
-------------
Name:       OneWayFreqs
Label:      One-Way Frequencies
Template:   Base.Freq.OneWayFreqs
Path:       Freq.Table1.OneWayFreqs
-------------

Output Added:
-------------
Name:       OneWayChiSq
Label:      One-Way Chi-Square Test
Template:   Base.Freq.StatFactoid
Path:       Freq.Table1.OneWayChiSq
-------------
NOTE: There were 500 observations read from the data set WORK.SALES.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

522  ODS TRACE OFF;
```

If you wanted, you could simply turn TRACE on and leave it on for the duration of your SAS session. Then all of your ODS output would be documented in the log. In practice, you'll find that you might need to run TRACE from time to time as you get to know each procedure and its output, but you might not want it on all of the time.

Identifying Output Objects in Your Listing

To get to know ODS output, you will find it helpful to turn ODS TRACE on at least the first time you run each procedure. After a while, you'll get to know the common procedures, and you might not need to trace your output every time. You can also look up output object names in SAS documentation.

Another way to run TRACE for a new procedure is to turn on the LISTING option (with the LISTING destination open). This tells TRACE to display its information in your listing output, instead of in the SAS log. The syntax is as follows:

```
      ODS TRACE ON / LISTING;
proc reg data=sales;
   model itemprice=itemssold department;
quit;
ODS TRACE OFF;
```

You get the results shown in Figure 6.3. All the output objects are visible in the Results window, and the first three objects and their TRACE information are shown in the listing output. Before each piece of the PROC REG output is a set of tracing information that names the output object, its table definitions, and its path. This makes it easy to see which names go to what parts of the output. Of course, it also makes your results a bit hard to read, so you'll want to rerun the procedure later with TRACE set to OFF.

In this example, the procedure creates four output objects, visible in the Results window. The first object is called NObs, or Number of Observations, representing the number of observations read and used. The second object is ANOVA, or Analysis of Variance, and it includes the analysis of variance results. The next output object, FitStatistics, contains a number of statistics that measure the fit of the model. The last output object, ParameterEstimates, contains the parameter estimates for each term in the model. Note that sometimes the name and the label are almost the same, but it is possible for them to be different (as shown with NObs and Number of Observations).

Every time you run a PROC REG with the same option settings and type of model, you will get these same output objects. Different PROC REG options or a different model can result in different object names. Not only are the output object names consistent across runs of the same procedure, they're also consistent across similar results from different procedures. If you run another model that generates analysis of variance results, you will get an output object named ANOVA. Similarly, you'll get your fit statistics in an output object named FitStatistics.

For example, if you run a PROC GLM with the same model as the PROC REG above, you get six output objects: NObs, OverallANOVA, FitStatistics, Type I ModelANOVA, Type III ModelANOVA, and ~~ParameterEstimates~~Solution, shown in Figure 6.4. Notice that the FitStatistics object has the same object name as the PROC REG object name, but a different path. We explore this difference in the discussion of identifying objects by full PATH name.

SAS procedures build output objects in a modular fashion. Because many procedures calculate Fit Statistics, this statistical output object is built from a common set of building blocks, contained in the table template definition for each procedure. Some procedures share object definitions. Of course, some procedures generate highly specialized output objects, so some output object definitions and names are used in only one procedure.

Figure 6.3

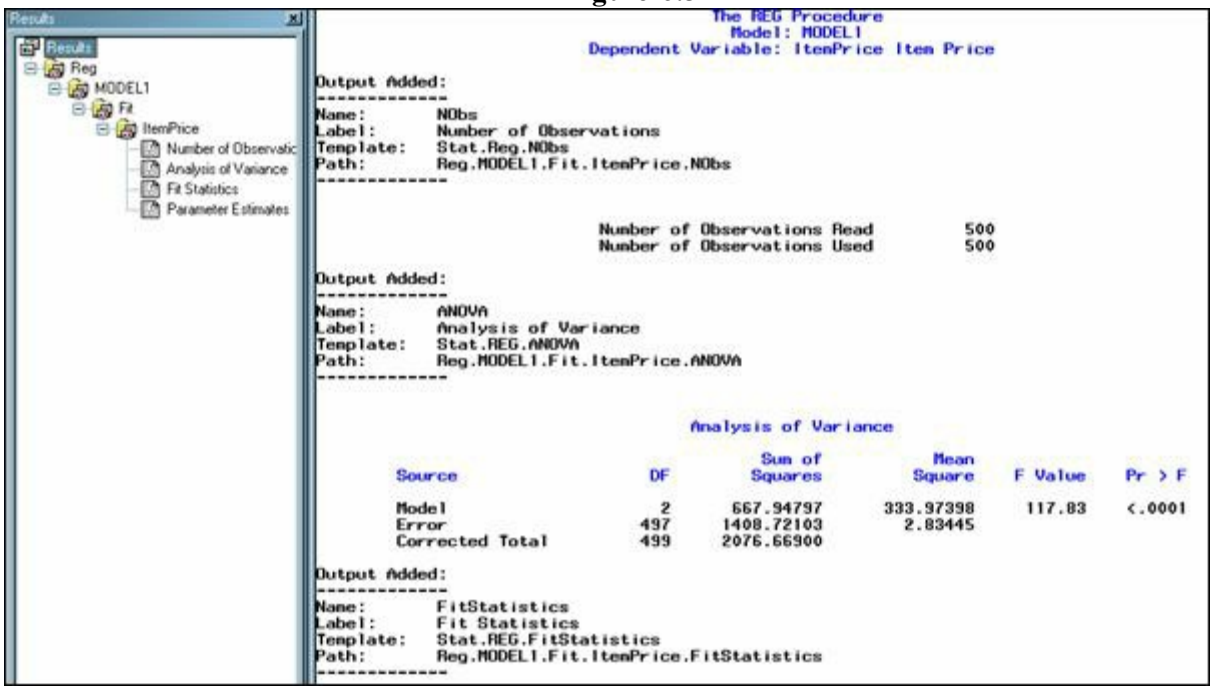
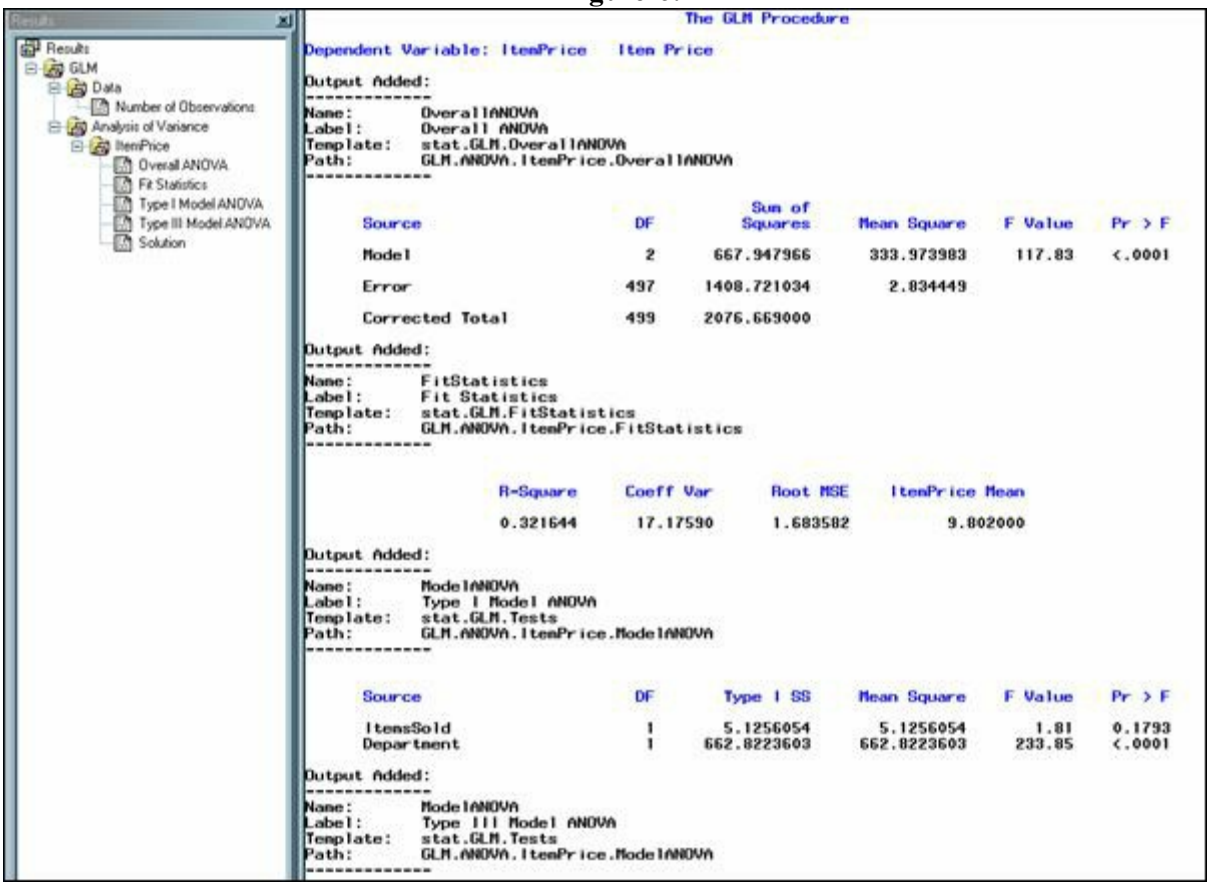


Figure 6.4



Identifying Output Objects by Label Path

The previous example showed how to find the names associated with each output object. When all of your output objects have unique names, this is the simplest way to identify each object. However, sometimes an output object name is not unique. For example, the GLM output shown in Figure 6.4 created three different types of ANOVA objects. Figure 6.5 shows that the Type I ANOVA and the Type III ANOVA both have the same object names, **ModelANOVA,** and both have the same pathname: **GLM.ANOVA.ItemPrice.ModelANOVA**. Only the object label differs for these two objects.

The way to see the most complete TRACE information is to turn on the LABEL option when you invoke ODS TRACE. Although you can put the label information into the SAS listing output, in the code example below, the output goes to the SAS log.

```
      ODS TRACE ON / LABEL;
proc glm data=sales;
  model itemprice=itemsold department;
run;
quit;
ODS TRACE OFF;
```

Figure 6.6 shows the traced results of PROC GLM using the LABEL option. Notice how the name, path, and template1 are the same for both ANOVA objects. Only the label is different. After you turn on the LABEL option, you see that there is another form of the object path information called the "Label Path," and as shown in Figure 6.6, the label path for the two ANOVA objects is different. If you wanted to select only the Type III object from PROC GLM, you'd have to identify it by the label path. For other procedures, it might be sufficient to identify the object by its name, simple path or some combination of name, path, and label path.

So why aren't all of the identifiers unique? Why do the two tables have the same name and path? The reason is that sometimes it's useful to be able to identify them as a group.

For example, if you wanted to change the table definition for ModelANOVA to display fewer decimal places for the mean squares, you would probably want to change both tables. The output would look odd if only one table was modified. By changing the table definition for Stat.GLM.Tests, you can affect both tables at the same time.

On the other hand, if you wanted to write only the Type III statistics to an output data set, the best way to do that is by using the label path. If you wanted to send the FitStatistics from PROC GLM to the RTF destination and the FitStatistics from PROC REG to the HTML destination, then using the pathname would be sufficient.

There are more ways to identify a unique output object name for ODS SELECT, ODS EXCLUDE, and ODS OUTPUT usage, as well as for ODS DOCUMENT usage. These

methods are discussed in Chapters 8 and 11.

Figure 6.5: ODS TRACE ON /LISTING Results

```
Output Added:
-------------
Name:        ModelANOVA
Label:       Type I Model ANOVA
Template:    stat.GLM.Tests
Path:        GLM.ANOVA.ItemPrice.ModelANOVA
-------------

      Source                      DF        Type I SS     Mean Square    F Value    Pr > F

      ItemsSold                    1        5.1256054       5.1256054       1.81    0.1793
      Department                   1      662.8223603     662.8223603     233.85    <.0001

Output Added:
-------------
Name:        ModelANOVA
Label:       Type III Model ANOVA
Template:    stat.GLM.Tests
Path:        GLM.ANOVA.ItemPrice.ModelANOVA
-------------

      Source                      DF       Type III SS     Mean Square    F Value    Pr > F

      ItemsSold                    1        5.8044606       5.8044606       2.05    0.1531
      Department                   1      662.8223603     662.8223603     233.85    <.0001
```

Figure 6.6: ODS TRACE ON / LABEL Results

```
Output Added:
-------------
Name:        ModelANOVA
Label:       Type I Model ANOVA
Template:    stat.GLM.Tests
Path:        GLM.ANOVA.ItemPrice.ModelANOVA
Label Path:  'The GLM Procedure','Analysis of Variance','ItemPrice','Type I Model ANOVA'
-------------

Output Added:
-------------
Name:        ModelANOVA
Label:       Type III Model ANOVA
Template:    stat.GLM.Tests
Path:        GLM.ANOVA.ItemPrice.ModelANOVA
Label Path:  'The GLM Procedure','Analysis of Variance','ItemPrice','Type III Model ANOVA'
-------------
```

In SAS 8, there were instances where the output object path generated a unique object name because it contained the variable name in the short path. This is still true for some objects. However, as seen with the ANOVA objects, in some instances, the object path is not detailed enough to distinguish the objects. In this case, it is likely that the label path will identify the object at the finest level of granularity.

Viewing the Results

If you run SAS interactively, then in addition to using the ODS TRACE statement, you can also view your output objects in the Results window. The Results window is a tree view of the output created during your current SAS session.

The top level of the tree is a SAS procedure or DATA step. Beneath the procedure are items for each of the output objects that the procedure creates. If you click on a procedure or an output object label in the Results window, it brings up the Output window and jumps to the ODS output that was created from that output object (assuming that you have ODS LISTING turned on so that there is output in the Output window).

To illustrate this, look at the following example:

```
      ODS TRACE ON / LABEL;
proc univariate data=sales;
   var ItemPrice;
run;
ODS TRACE OFF;
```

The Results window is shown in Figure 6.7. The log file with the TRACE information is shown in Figure 6.8.

In the Results window, the tree has an item for the UNIVARIATE procedure, and then items below that for each of the output objects. Notice how the names of the items in the tree —Moments, Basic Measures of Location, Variability, etc.—match up with the labels shown in the TRACE output.

You can use a label to identify an output object in place of the name or path of the object. Because the labels are used in the Results window, sometimes this is a handy way to identify output objects. You don't even need to run a TRACE, because the labels are right there in the Results window. Just remember that labels are generally specified as quoted strings in ODS SELECT, ODS EXCLUDE, or ODS OUTPUT statements.

Figure 6.7

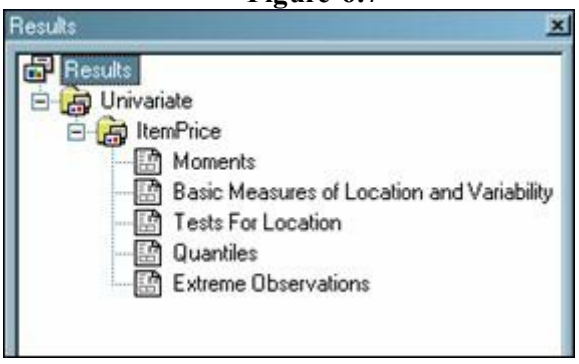


Figure 6.8

```
Output Added:
-------------
Name:        Moments
Label:       Moments
Template:    base.univariate.Moments
Path:        Univariate.ItemPrice.Moments
Label Path:  'The Univariate Procedure'.'ItemPrice'.'Moments'
-------------

Output Added:
-------------
Name:        BasicMeasures
Label:       Basic Measures of Location and Variability
Template:    base.univariate.Measures
Path:        Univariate.ItemPrice.BasicMeasures
Label Path:  'The Univariate Procedure'.'ItemPrice'.'Basic Measures of Location and Variability'
-------------

Output Added:
-------------
Name:        TestsForLocation
Label:       Tests For Location
Template:    base.univariate.Location
Path:        Univariate.ItemPrice.TestsForLocation
Label Path:  'The Univariate Procedure'.'ItemPrice'.'Tests For Location'
-------------

Output Added:
-------------
Name:        Quantiles
Label:       Quantiles
Template:    base.univariate.Quantiles
Path:        Univariate.ItemPrice.Quantiles
Label Path:  'The Univariate Procedure'.'ItemPrice'.'Quantiles'
-------------

Output Added:
-------------
Name:        ExtremeObs
Label:       Extreme Observations
Template:    base.univariate.ExtObs
Path:        Univariate.ItemPrice.ExtremeObs
Label Path:  'The Univariate Procedure'.'ItemPrice'.'Extreme Observations'
-------------
NOTE: PROCEDURE UNIVARIATE used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds
```

TRACE Examples: Other Procedures

Although the previous examples have emphasized the similarities in ODS output objects across procedures, there are some procedures that are unique. For example, PROC CHART uses monospace fonts to produce low-resolution graphics, whereas PROC GCHART uses the new ODS Graphics framework in SAS 9.2 to produce high-resolution graphics.

```
      ODS TRACE ON;
proc chart data=sales;
   vbar ProductName;
run;
proc gchart data=sales;
   vbar ProductName;
quit;
ODS TRACE OFF;
```

As you can see in Figure 6.9, these procedures each produce a single output object, but each output object reflects the nature of the output. From PROC CHART, the object is called VBAR, whereas for PROC GCHART, the object is called GCHART. In addition, both of these output objects have a property called “Data Name” that distinguishes between the batch output from PROC CHART and the GRSEG catalog storage of the data points for PROC GCHART.

Another interesting procedure is PROC CONTENTS, which produces a number of output objects that are unique to this procedure. Figure 6.10 shows the SAS log results for this PROC CONTENTS step.

```
      ODS TRACE ON / LABEL;
proc contents data=sales;
run;
ODS TRACE OFF;
```

Figure 6.9

```
749  ODS TRACE ON;
750  proc chart data=sales;
751     vbar ProductName;
752  run;


Output Added:
-------------
Name:        VBAR
Label:       VBAR of ProductName
Data Name:   BatchOutput
Path:        Chart.VBAR
-------------
NOTE: PROCEDURE CHART used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


753  proc gchart data=sales;
754     vbar ProductName;
755  run;


Output Added:
-------------
Name:        GCHART3
Label:       Bar chart of ProductName
Data Name:   GRSEG
Path:        Gchart.GCHART3
-------------
756  quit;
```

Figure 6.10

```
Output Added:
-------------
Name:        Attributes
Label:       Attributes
Template:    Base.Contents.Attributes
Path:        Contents.DataSet.Attributes
Label Path:  'The Contents Procedure'.'WORK.SALES'.'Attributes'
-------------

Output Added:
-------------
Name:        EngineHost
Label:       Engine/Host Information
Template:    Base.Contents.EngineHost
Path:        Contents.DataSet.EngineHost
Label Path:  'The Contents Procedure'.'WORK.SALES'.'Engine/Host Information'
-------------

Output Added:
-------------
Name:        Variables
Label:       Variables
Template:    Base.Contents.Variables
Path:        Contents.DataSet.Variables
Label Path:  'The Contents Procedure'.'WORK.SALES'.'Variables'
-------------
```

TRACE Examples: Procedures without ODS Output

Some procedures produce ODS output only in certain conditions. Generally, PROC FORMAT is used to create and store formats. However, it can produce output. In the following example, the FMTLIB option is used to generate a listing of the formats stored in the format library called LIBRARY. The output object unique to PROC FORMAT is shown in Figure 6.11.

```
      ODS TRACE ON / LABEL;
proc format library=library fmtlib;
run;
ODS TRACE OFF;
```

Finally, some procedures don't produce ODS output at all. In this next example, we look at PROC SORT, PROC PRINTTO, and PROC MEANS. Though PROC SORT and PROC PRINTTO do produce output, they do not produce any visible output in the LISTING destination, so there are no ODS output objects to trace.

In the following example, the MEANS procedure generates an ODS output object, BASE.SUMMARY, but the SORT and PRINTTO procedures do not. The TRACE information, and lack of TRACE information, is shown in Figure 6.12.

```
      ODS TRACE ON / LABEL;
proc sort data=sales out=sales;
  by Department;
run;
proc printto print=printf new;

      run;
proc means data=sales;
   var ItemsSold;
run;
proc printto; run;
ODS TRACE OFF;
```

It is also possible to trace any excluded output objects. This advanced technique is shown in Chapter 8, "Limiting ODS Output."

Figure 6.11

```
Output Added:
-------------
Name:        FORMAT
Label:       FMTLIB Listing
Data Name:   BatchOutput
Path:        Format.FORMAT
Label Path:  'The Format Procedure'.'FMTLIB Listing'
-------------
```

Figure 6.12

```
780  ODS TRACE ON / LABEL;
781  proc sort data=sales out=sales;
782    by Department;
783  run;

NOTE: There were 500 observations read from the data set WORK.SALES.
NOTE: The data set WORK.SALES has 500 observations and 8 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.18 seconds
      cpu time            0.00 seconds


784  proc printto print=printf new;
785  run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


786  proc means data=sales;
787     var ItemsSold;
788  run;


Output Added:
-------------
Name:       Summary
Label:      Summary statistics
Template:   base.summary
Path:       Means.Summary
Label Path: 'The Means Procedure'.'Summary statistics'
-------------
NOTE: There were 500 observations read from the data set WORK.SALES.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.06 seconds
      cpu time            0.04 seconds


789  proc printto; run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds


790  ODS TRACE OFF;
```

Chapter 7
Output Data Sets

You can use ODS to create complete output data sets from almost every SAS procedure. These output data sets contain virtually every bit of information that you see in the listing results. In addition, starting with SAS 9.2, you have greater control over how you select the ODS objects for output to report destinations and data sets.

This chapter provides a number of examples from different procedures to give you an overview of output data sets.

This chapter uses the data sets TRAFFIC and PRODUCT to illustrate the concepts. These data sets are available on the author pages for this book at http://support.sas.com/authors.

Creating a Single Output Data Set

The first thing you need to know about ODS output data sets is that you can create a lot of them. A single procedure frequently generates multiple output objects, and you can create data sets from each of the output objects produced by a procedure.

Before covering the complexities of handling these multiple output data sets, we're going to start out simple. This first example looks at creating just one output data set. To request an output data set, you use the following ODS statement:

```
ODS OUTPUT output-object=data-set-name;
```

Also, you can specify more than one output object and data set name pair in your invocation. The syntax would look like this:

```
ODS OUTPUT output-object=data-set-name
   output-object-n=data-set-name-n;
```

If you issue the ODS OUTPUT statement by itself, there is no default output data set produced. Specifying an output object is required.

In this example, we create an output data set with the results from a PROC MEANS step. Here is the code to do this:

```
      ODS OUTPUT "Summary statistics"=Stats;
proc means data=Traffic;
   var MPH;
run;
ODS OUTPUT CLOSE;
```

The output object we want is labeled "Summary statistics" and it will be routed to the output data set called STATS. To identify this object, you can submit an ODS TRACE ON statement. Another option is to run the procedure once and then look at the label in the Results window.

When you run a PROC MEANS step, the Results window displays a folder labeled "Means" as shown in Figure 7.1. If you drill down to the next level, you see the label "Summary statistics." These labels at the most detailed level of the Results window are the output object labels that you need for ODS OUTPUT.

When the preceding ODS OUTPUT code is submitted, the PROC MEANS results are displayed in the Output window, and a data set called STATS is generated. You can then print the data set to see what it looks like. To illustrate this example, RTF output generated from the PROC MEANS step is displayed in Figure 7.2. A PROC PRINT listing of the contents of the data set STATS is displayed in Figure 7.3. By comparing the procedure output and the output data set, you can see how the two relate.

Each of the resulting statistics from the PROC MEANS step is included in the output data

set. Each of the output variables has a name that indicates the original variable and the statistic it contains. As you begin using ODS output data sets, you will find it helps to explore the data in order to learn the relationships between the procedure output and the output data set structures.

Figure 7.1

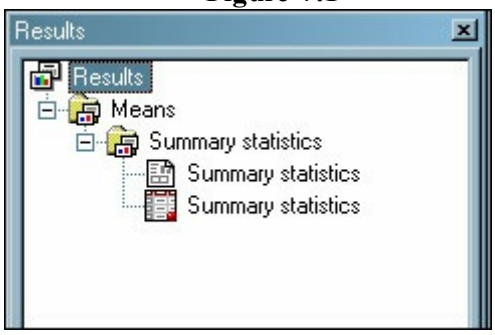


Figure 7.2: Output from the MEANS Procedure

The MEANS Procedure

Analysis Variable : MPH Average Speed				
N	Mean	Std Dev	Minimum	Maximum
189	27.8756614	18.0832631	0	65.0000000

Figure 7.3: PROC PRINT Output for Data Set from the PROC MEANS Step

Obs	MPH_N	MPH_Mean	MPH_StdDev	MPH_Min	MPH_Max
1	189	27.875661376	18.083263104	0	65

There is one risk to selecting output objects by their labels instead of their names. If your code will be run by users in regions with different languages, the label will not be the same in their version of SAS. However, the names of output objects are the same, no matter which language version of SAS is used.

Comparing ODS Output Data Sets to Printed Output

To further explore the relationship between procedure results and output data sets, let's look at a more complex procedure. The previous example generated only one output data set. This time we'll look at a procedure that creates two.

Figure 7.4 shows the RTF results of a PROC CORR step. If you look carefully at this RTF output, you can see that there are two tables of results. The first table describes the two variables, and the second table displays the correlation coefficients.

These are two separate output objects. To see this, you can look at the Results window displayed in Figure 7.5. It shows two items under the Corr folder. One object is labeled "Variables Information" and the other is labeled "Pearson Correlations."

The output object we are going to use for this example is the one that contains the main results, the correlation coefficients. Here is the code to request an output data set for this object:

```
      ODS OUTPUT "Pearson Correlations"=PearsonCorr;
proc corr data=Traffic nosimple;
  var TimeofDay;
  with MPH;
run;
ODS OUTPUT CLOSE;
```

When the preceding code is submitted, the correlation results are displayed in the Output window, and a data set called PEARSONCORR is generated. This data set is shown in Figure 7.6 using Viewtable in SAS Explorer. Each of the pieces of information in the correlation matrix is included in the output data set. However, the format is different. The correlation matrix has a crosstabulation format, but the output data set has just one record. The two results have been included as two variables: TIMEOFDAY is the correlation coefficient, and PTIMEOFDAY is the *p*-value.

If we wanted to create a data set from the other output object, the table of variable information, our code would be modified as follows:

```
      ODS OUTPUT "Variables Information"=VarInfo;
proc corr data=Traffic nosimple;
  var TimeofDay;
  with MPH;
run;
ODS OUTPUT CLOSE;
```

When the modified code is submitted, the correlation results are displayed in the Output window, and a data set called VARINFO is generated. Figure 7.7 shows this data set. Each of the pieces of information in the variable information table is included in the output data set. This time, the structure is the same in both the printed results (Figure 7.4) and the output data set (Figure 7.7). Both have two rows of information. ODS has created one record for each

variable in the variables table.

Figure 7.4

1 With Variables:	MPH
1 Variables:	TimeofDay

Pearson Correlation Coefficients, N = 189 Prob > |r| under H0: Rho=0	
	TimeofDay
MPH Average Speed	-0.03555 0.6272

Figure 7.5

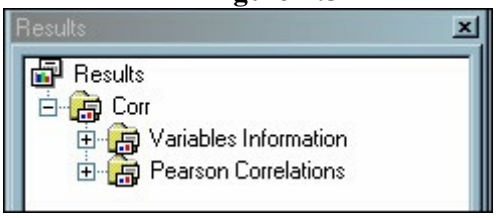
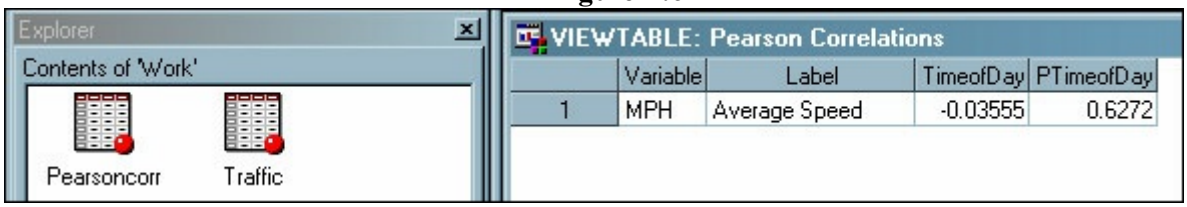


Figure 7.6

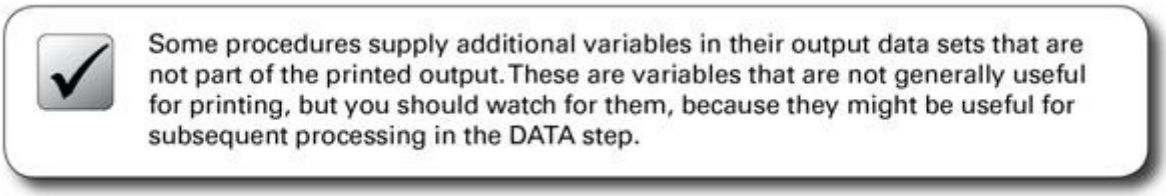


	Variable	Label	TimeofDay	PTimeofDay
1	MPH	Average Speed	-0.03555	0.6272

Figure 7.7

VIEWTABLE: Variables Information

	NVars	Variables	VarNames
1	1	With Variables:	MPH
2	1	Variables:	TimeofDay



Creating Multiple Output Data Sets from a Single Procedure

In the previous example, we used two separate PROC CORR steps to create two output data sets. However, if you want two output data sets from the same procedure, you don't have to run the procedure twice. You can just request two different output data sets from the same step.

For example, if we run a PROC FREQ and generate not just a frequency distribution but also a series of chi-square statistics, we can create output data sets for both results with just one PROC FREQ step. On the left side of Figure 7.8, the Results window shows the two output objects created by the PROC FREQ step: Cross-Tabular Freq Table and Chi-Square Tests. The right side of Figure 7.8 shows the Output window with the output from the PROC FREQ step. The following code uses two ODS OUTPUT statements before the procedure code to request two data sets, and it uses a single ODS OUTPUT CLOSE statement to end data set creation.

```
      ODS OUTPUT "Cross-Tabular Freq Table"=FreqTable;
ODS OUTPUT "Chi-Square Tests"=FreqChiSq;
proc freq data=Traffic;
  tables highway*event / chisq norow nocol nopct;
run;
ODS OUTPUT CLOSE;
```

A simpler way to write this syntax is to put both data set requests in a single ODS OUTPUT statement. The following code is equivalent to the preceding code.

```
      ODS OUTPUT "Cross-Tabular Freq Table"=FreqTable
         "Chi-Square Tests"=FreqChiSq;
proc freq data=Traffic;
  tables highway*event / chisq norow nocol nopct;
run;
ODS OUTPUT CLOSE;
```

Everything is the same for the two versions of the program. Viewtable displays the resulting data sets in Figure 7.9 and Figure 7.10. Notice that the structures of the two data sets are very different. The first data set, FREQTABLE, has detailed data on observation counts for the two variables, and it has one record for each combination of the two variables. The second data set, FREQCHISQ, has summary statistics and has one record for each statistic.

This is why ODS OUTPUT produces multiple data sets. If you tried to create a single data set from all of the different types of output objects produced by a procedure, you'd end up with very confusing results.

Sometimes what you get in your output data set is not what you expected. For example, you might think that the output data set from a PROC FREQ step would have the same shape as the table: rows for the row variable (HIGHWAY) and columns for the column variable (EVENT). Instead, PROC FREQ output data sets have a row for every combination of the row and column variables, plus rows at the end for the totals. This is a common format for ODS output data sets when the procedure creates crosstabular results. So, before you assume that what you see (in the Output window) is what you get (from Viewtable), take the time to print and review your ODS data sets.

Figure 7.8

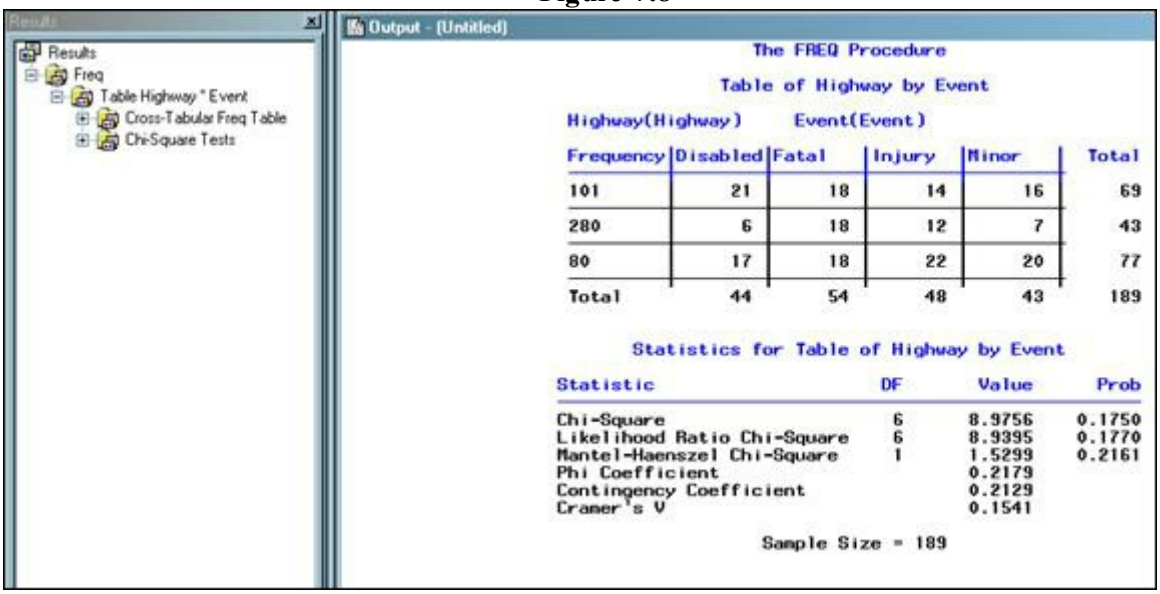


Figure 7.9: Work.FreqTable

VIEWTABLE: Frequency Counts and Percentages

	Table	Highway	Event	Variables Contributing to Observation	Table Number	Frequency Count	Frequency Missing
1	Table Highway * Event	101	Disabled	11	1	21	.
2	Table Highway * Event	101	Fatal	11	1	18	.
3	Table Highway * Event	101	Injury	11	1	14	.
4	Table Highway * Event	101	Minor	11	1	16	.
5	Table Highway * Event	101		10	1	69	.
6	Table Highway * Event	280	Disabled	11	1	6	.
7	Table Highway * Event	280	Fatal	11	1	18	.
8	Table Highway * Event	280	Injury	11	1	12	.
9	Table Highway * Event	280	Minor	11	1	7	.
10	Table Highway * Event	280		10	1	43	.
11	Table Highway * Event	80	Disabled	11	1	17	.
12	Table Highway * Event	80	Fatal	11	1	18	.
13	Table Highway * Event	80	Injury	11	1	22	.
14	Table Highway * Event	80	Minor	11	1	20	.
15	Table Highway * Event	80		10	1	77	.
16	Table Highway * Event		Disabled	01	1	44	.
17	Table Highway * Event		Fatal	01	1	54	.
18	Table Highway * Event		Injury	01	1	48	.
19	Table Highway * Event		Minor	01	1	43	.
20	Table Highway * Event			00	1	189	0

Figure 7.10: Work.FreqChiSq

VIEWTABLE: Chi-Square Tests

	Table	Statistic	DF	Value	Prob
1	Table Highway * Event	Chi-Square	6	8.9756	0.1750
2	Table Highway * Event	Likelihood Ratio Chi-Square	6	8.9395	0.1770
3	Table Highway * Event	Mantel-Haenszel Chi-Square	1	1.5299	0.2161
4	Table Highway * Event	Phi Coefficient	_	0.2179	_
5	Table Highway * Event	Contingency Coefficient	_	0.2129	_
6	Table Highway * Event	Cramer's V	_	0.1541	_

Creating Output Data Sets from Procedures with Complex Output

In the next example, we explore how a single procedure creates 15 output objects! It's not a complicated procedure, just a simple PROC UNIVARIATE with a CLASS variable. The output structure is shown in Figure 7.11.

The reason we get so many output objects is the CLASS variable HIGHWAY. Without a CLASS variable, this procedure produces five output objects: Moments, Basic Measures of Location and Variability, Tests for Location, Quantiles, and Extreme Observations. Each of these five objects is created for each of the three values for the HIGHWAY CLASS variable.

If we want just one of the output objects, we can request an output data set that contains that object for each of the three values of the CLASS variable:

```
      ODS OUTPUT "Moments"=UnivMPH;
proc univariate data=Traffic;
  class Highway;
  var MPH;
run;
ODS OUTPUT CLOSE;
```

This code creates a single data set called UNIVMPH that contains the results for all three highways. Viewtable displays this data set in Figure 7.12. This single data set contains three output objects. This works because all three objects have the same label: "Moments."

To identify each of the three output objects, you can look at the value of the variable HIGHWAY. Notice how the first six rows in the data set are for HIGHWAY=101, which is the first of the three class groups, and the first output object. The next six rows are for HIGHWAY=280, and the final six rows are for HIGHWAY=80. Every row has the same variables, as output by PROC UNIVARIATE. Every CLASS variable, however, has its own group of rows in the output data set.

Be careful when combining data sets from different output objects. Just because they look similar and come from the same procedure does not mean that their formats are identical. You might have a variable that is shorter in one data set than in another, which will cause it to be truncated when you combine the data sets.

Figure 7.11

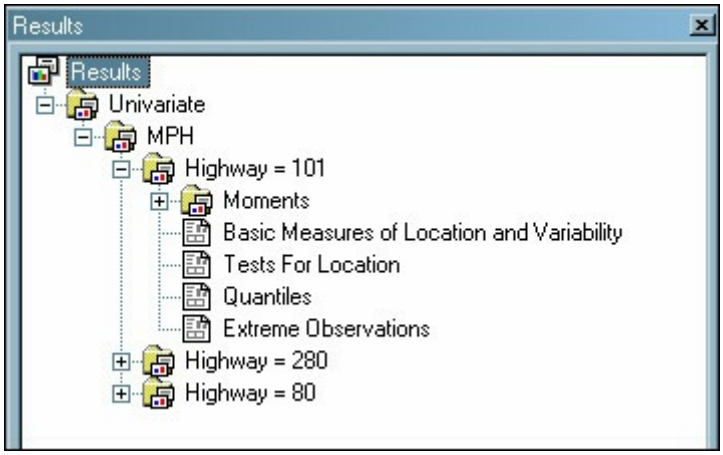


Figure 7.12

VIEWTABLE: Moments

	VarName	Highway	Label1	cValue1	nValue1	Label2	cValue2	nValue2
1	MPH	101	N	69	69.000000	Sum Weights	69	69.000000
2	MPH	101	Mean	27.4565217	27.456522	Sum Observations	1894.5	1894.500000
3	MPH	101	Std Deviation	17.1055846	17.105585	Variance	292.601023	292.601023
4	MPH	101	Skewness	0.69307843	0.693078	Kurtosis	-0.5391733	-0.539173
5	MPH	101	Uncorrected SS	71913.25	71913	Corrected SS	19896.8696	19897
6	MPH	101	Coeff Variation	62.3006247	62.300625	Std Error Mean	2.05927039	2.059270
7	MPH	280	N	43	43.000000	Sum Weights	43	43.000000
8	MPH	280	Mean	30.5465116	30.546512	Sum Observations	1313.5	1313.500000
9	MPH	280	Std Deviation	19.9254007	19.925401	Variance	397.021595	397.021595
10	MPH	280	Skewness	0.37027248	0.370272	Kurtosis	-1.0003976	-1.000398
11	MPH	280	Uncorrected SS	56797.75	56798	Corrected SS	16674.907	16675
12	MPH	280	Coeff Variation	65.2297093	65.229709	Std Error Mean	3.03859513	3.038595
13	MPH	80	N	77	77.000000	Sum Weights	77	77.000000
14	MPH	80	Mean	26.7597403	26.759740	Sum Observations	2060.5	2060.500000
15	MPH	80	Std Deviation	17.9510684	17.951068	Variance	322.240858	322.240858
16	MPH	80	Skewness	0.47199941	0.471999	Kurtosis	-0.8409398	-0.840940
17	MPH	80	Uncorrected SS	79628.75	79629	Corrected SS	24490.3052	24490
18	MPH	80	Coeff Variation	67.0823717	67.082372	Std Error Mean	2.04571411	2.045714

Creating Output Data Sets from Multiple Procedures: Part 1

Not only does ODS enable you to combine output objects from a single procedure into a common data set, but you can also combine same-named output objects from multiple procedures into a common data set. If you are running a single procedure repeatedly, it can be useful to be able to create a combined data set.

For example, let's say you are running a series of models to explore the relationships between your variables. This can generate pages and pages of printed output. It would be much nicer to be able to produce a simple summary table. This example looks at how to combine the odds ratio results from a series of PROC LOGISTIC runs. After running the procedure once as a test, we can see from the Results window in Figure 7.13 that the output object we want is called Odds Ratios. The following code asks that an output data set be created from this object.

```
      ODS OUTPUT "Odds Ratios"=FatalOdds;
proc logistic data=Traffic;
  model Fatal=MPH;
run;
      proc logistic data=traffic;
        class Location;
  model Fatal=Location;
run;
proc logistic data=traffic;
  class Location Highway;
  model Fatal=Location Highway;
run;
ODS OUTPUT CLOSE;
```

The resulting data set is shown in Figure 7.14. It contains the odds ratios we want, but something is wrong. It contains only the results from the first model. This is because ODS only holds the output data set open for receiving data until the next PROC step or DATA step starts.

To keep the definition in place across multiple procedures, we need to add an option to the ODS OUTPUT statement. Turn the page to see how this is done.

Figure 7.13

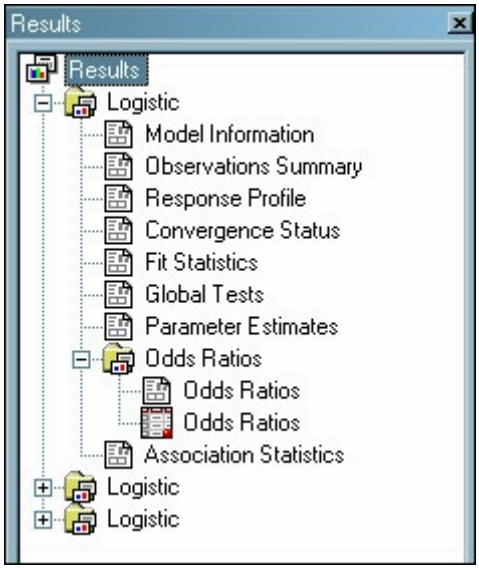


Figure 7.14

VIEWTABLE: Odds Ratios

	Effect	Odds Ratio Estimate	Lower 95% Confidence Limit for Odds Ratio	Upper 95% Confidence Limit for Odds Ratio
1	MPH	1.004	0.986	1.022

Creating Output Data Sets from Multiple Procedures: Part 2

To keep the definition in place across multiple procedures, we need to add an option to the ODS OUTPUT statement.

The PERSIST= option in the ODS OUTPUT statement tells ODS to concatenate the data sets from *all* the procedures in the program until an ODS OUTPUT CLOSE statement executes. The ODS OUTPUT statement is specified as follows. The data set created is called ALLODDS.

```
ODS OUTPUT "Odds Ratios"(PERSIST=PROC)=AllOdds;
```

Viewtable displays the ALLODDS data set in Figure 7.15.

The final data set will have variables from all the concatenated data sets. Variables defined in one output object but not defined in a second output object will have missing values for those variables in the observations contributed from the second data set. If a variable created by an output object has a different type than a variable with the same name created by an earlier output object, the variable from the later output object is assigned a different name formed by adding a numeric suffix.

Before SAS 9.1.3, you needed to add another option, MATCH_ALL. This option tells ODS to create a new data set for each of the multiple output objects. For more information about its usage, see the SAS OnlineDoc.

Figure 7.15

VIEWTABLE: Odds Ratios

	Proc Name	Run Group Numbe	Effect	Odds Ratio Estimate	Lower 95% Confidence Limit for Odds Ratio	Upper 95% Confidence Limit for Odds Ratio
1	Logistic	1	MPH	1.004	0.986	1.022
2	Logistic	1	Location East Bay vs South Bay	0.390	0.089	1.715
3	Logistic	1	Location San Francisco vs South Bay	0.288	0.082	1.015
4	Logistic	1	Location East Bay vs South Bay	0.335	0.069	1.632
5	Logistic	1	Location San Francisco vs South Bay	0.354	0.097	1.285
6	Logistic	1	Highway 101 vs 80	0.783	0.329	1.865
7	Logistic	1	Highway 280 vs 80	0.460	0.184	1.153

This example used PERSIST=PROC to create a file for each procedure. You can also use PERSIST=RUN to create a file for each run group. This is useful when working with procedures such as GLM.

Creating Output Data Sets without Printed Output

When you're running a procedure to produce an output data set, you might want to get rid of the results displayed in the Output window or listing file. The NOPRINT option, in addition to suppressing output from several procedures, can also prevent creation of ODS output data sets.

For example, let's say we're running a single model with the LOGISTIC procedure and we want to build a data set with the parameter estimates. After running the code once to find out the name of the output object (Parameter Estimates, as shown in Figure 7.16), you can set up the code as follows:

```
      ODS OUTPUT "Parameter Estimates"=Params;
proc logistic=traffic NOPRINT;
   model fatal=mph timeofday;
quit;
ODS OUTPUT CLOSE;
```

The NOPRINT option has been turned on to eliminate the unnecessary listing output, since the data set PARAMS will hold the results we need. Or does it? If you view Figure 7.17, you can see from the warning message in the SAS log that the output data set PARAMS was never created. This is because in order for ODS to create an output data set (or HTML file or RTF file or Printer output), there has to be an output object. With the NOPRINT procedure option turned on, the output object can't be created.

To get rid of the unwanted listing output *and* to create an ODS output data set, you need to use ODS instead of the NOPRINT option. The listing output is just another form of ODS output, which can be turned on and off. By default, the LISTING destination is turned on. If you don't want listing output, just turn it off with the ODS LISTING CLOSE statement.

Our program above is revised below to create an output data set without creating listing output.

```
      ODS LISTING CLOSE;
ODS OUTPUT "Parameter Estimates"=Params;
proc logistic=traffic;
   model fatal=mph timeofday;
quit;
ODS OUTPUT CLOSE;
ODS LISTING;
```

The ODS LISTING CLOSE statement shuts off the LISTING destination. Then the ODS OUPUT statement captures the output data set called PARAMS. Finally, another ODS LISTING statement turns the LISTING destination back on. This time there's no warning message in the SAS log because data set PARAMS is created. Viewtable displays its contents in Figure 7.18.

Figure 7.16

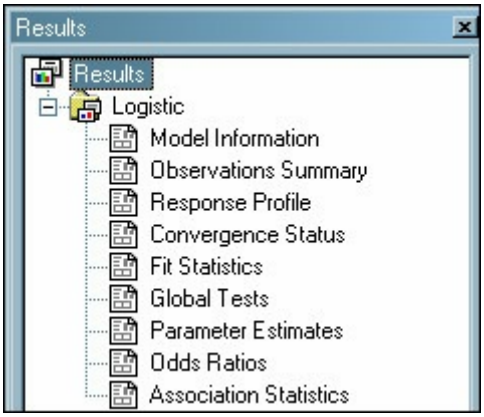


Figure 7.17

```
231  ODS OUTPUT "Parameter Estimates"=Params;
232  proc logistic data=traffic NOPRINT;
233     model fatal=mph timeofday;
234  quit;

NOTE: PROC LOGISTIC is modeling the probability that Fatal=0. One way to change this to model
      the probability that Fatal=1 is to specify the response variable option EVENT='1'.
NOTE: Convergence criterion (GCONV=1E-8) satisfied.
NOTE: There were 189 observations read from the data set WORK.TRAFFIC.
NOTE: PROCEDURE LOGISTIC used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

WARNING: Output ''Parameter Estimates'' was not created.  Make sure that the output object
         name, label, or path is spelled correctly.  Also, verify that the appropriate
         procedure options are used to produce the requested output object.  For example,
         verify that the NOPRINT option is not used.
```

Figure 7.18

VIEWTABLE: Parameter Estimates

	Variable	DF	Estimate	Standard Error	Wald Chi-Square	Pr > Chi-Square
1	Intercept	1	-0.0180	0.4122	0.0019	0.9652
2	MPH	1	0.00532	0.00915	0.3379	0.5610
3	TimeofDay	1	0.0697	0.0249	7.8488	0.0051

In addition to the TRACE statement and the Results window, you can also find output object names by reviewing your procedure's online documentation. SAS documentation includes the ODS output object names for each procedure.

Tip: Turning Off Output to the Results Window

One of the more common reasons for creating an output data set is to enable combining results from a large number of procedures. For example, it is common to run a statistical test on a large number of subgroups, and then produce a summary table of selected statistics.

The following example runs the TTEST procedure on five subsets of the data set PRODUCT defined by the values of PLANT. Macro program TESTIT is used to send the results to five output data sets. However, it could just as easily have created 50 or 200 output data sets if the upper index value in the %DO statement were modified and the data set contained that number of subsets. This macro program is a modification of the macro program seen in a previous tip.

```
    %macro testit;
  %do B=1 %to 5;
   ODS OUTPUT TTests=TTest&B;
   proc ttest data=product (where=(plant=&B));
    class product;
    var units;
   run;
   ODS OUTPUT CLOSE;
  %end;
%mend testit;
ODS LISTING CLOSE;
%testit;
```

The LISTING destination is closed to avoid sending all of the results to the Output window. However, the Results window is still filled with links to five results, as shown in Figure 7.19.

There are two ways to get rid of these unnecessary results. First, you could run the program in batch mode, where there will be no Results window.

The second option is to turn off the delivery of results to the Results window. This is done using the ODS RESULTS statement. It can be set to RESULTS=OFF or NORESULTS to stop results from being displayed in the Results window.

Running the following code still produces the same five output data sets, but the Results window remains blank, as shown in Figure 7.20. A partial view of the SAS log shows that the macro program did run and the output data sets were created, even though the Results window is empty.

```
    ODS RESULTS=OFF;
%testit;
```

To allow ODS to place items in the Results window again, turn delivery of ODS results back on:

```
    ODS RESULTS=ON;
```

Figure 7.19

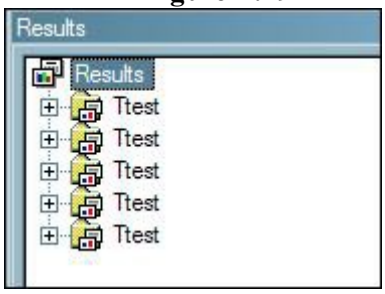


Figure 7.20

```
243
244
245  ODS RESULTS=OFF;
246  %testit;
MLOGIC(TESTIT):  Beginning execution.
MLOGIC(TESTIT):  %DO loop beginning; index variable B; start value is 1; stop value is 5; by
      value is 1.
MPRINT(TESTIT):   ODS OUTPUT TTests=TTest1;
MPRINT(TESTIT):   proc ttest data=product (where=(plant=1));
MPRINT(TESTIT):   class product;
MPRINT(TESTIT):   var units;
MPRINT(TESTIT):   run;

NOTE: The data set WORK.TTEST1 has 2 observations and 6 variables.
NOTE: PROCEDURE TTEST used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


MPRINT(TESTIT):   ODS OUTPUT CLOSE;
MLOGIC(TESTIT):  %DO loop index variable B is now 2; loop will iterate again.
MPRINT(TESTIT):   ODS OUTPUT TTests=TTest2;
MPRINT(TESTIT):   proc ttest data=product (where=(plant=2));
MPRINT(TESTIT):   class product;
MPRINT(TESTIT):   var units;
MPRINT(TESTIT):   run;

NOTE: The data set WORK.TTEST2 has 2 observations and 6 variables.
NOTE: PROCEDURE TTEST used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


MPRINT(TESTIT):   ODS OUTPUT CLOSE;
MLOGIC(TESTIT):  %DO loop index variable B is now 3; loop will iterate again.
```

Opening and Closing Output Data Sets

You might think that the results of every procedure between two ODS OUTPUT statements and an ODS OUTPUT CLOSE statement are sent to an output data set, but that's not quite true. After you open the ODS OUTPUT destination, it captures every specified object that is created before the next ODS OUTPUT CLOSE statement.

A complication can arise, however, if the ODS OUTPUT CLOSE statement occurs before SAS comes to another procedure step, DATA step, RUN statement, or QUIT statement. Without such a step boundary, the procedure is not finished when ODS closes the OUTPUT destination because the ODS OUTPUT statement executes immediately. In this situation, you might not get the ODS output data sets you expect.

The lack of a step boundary doesn't usually have this kind of impact on your results, so let's look at an example. This code runs one TTEST step, and this step does not end with a RUN statement. An ODS OUTPUT CLOSE statement follows the PROC TTEST statements and precedes the PROC PRINT step.

```
      ODS OUTPUT "T-Tests"=TTests;
proc ttest data=traffic;
  class fatal;
  var MPH;
ODS OUTPUT CLOSE;
      proc print data=TTests;
      run;
```

The Results window is shown in Figure 7.21. The SAS log from the above program is shown in Figure 7.22. It looks as though our code should create one data set, TTESTS, but it doesn't. The data set TTESTS is not created because SAS does not encounter a step boundary before it comes to the ODS OUTPUT CLOSE statement. The PROC PRINT statement following the ODS OUTPUT CLOSE statement forms the step boundary. By the time SAS reaches the PROC PRINT statement, it's too late to create the TTESTS output data set because the output destination is already closed.

To correct this, just insert a RUN or QUIT statement before the ODS OUTPUT CLOSE statement.

The problem illustrated in this example is not limited to ODS output data sets. You can run into the same problem with HTML, RTF, or printer output. Be sure to put a RUN or QUIT statement before every ODS CLOSE statement. Remember that ODS destination statements do not act as step boundaries.

Figure 7.21

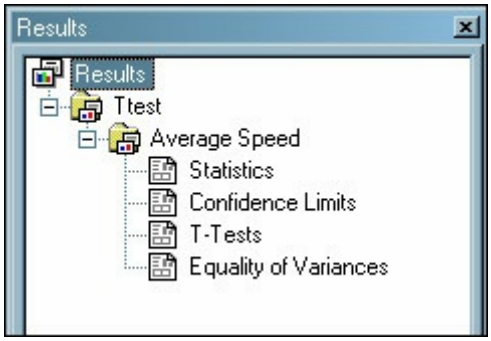


Figure 7.22

```
356
357  ODS OUTPUT "T-Tests"=TTests;
358  proc ttest data=traffic;
359     class fatal;
360     var MPH;
361  ODS OUTPUT CLOSE;

NOTE: PROCEDURE TTEST used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


362  proc print data=TTests;
ERROR: File WORK.TTESTS.DATA does not exist.
363  run;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Using Data Set Options with Output Data Sets

In each of the previous examples, we've selected specific output data sets. The data sets were chosen from the list of available output objects in the Results window. The contents of each of these data sets is determined by ODS.

For example, if you run the RSREG procedure and request a data set with the parameter estimates (see Figure 7.23), you automatically get the data shown in Figure 7.24.

```
      ODS OUTPUT "Parameter Estimates"=Params;
proc rsreg data=traffic;
  model fatal=mph;
run;
ODS OUTPUT CLOSE;
```

The data set PARAMS has eight variables. Every time you request this data set when running PROC RSREG, you get the same variables with the same names. However, you might not always want all eight variables, and you might not like their default names. For example, let's say we want only three of the variables, and you would like to rename PROBT to PVALUE. One way to achieve this would be to write a DATA step like the following:

```
      data Params2;
  set Params (keep=Parameter Estimate Probt
             rename=(Probt=Pvalue));
run;
```

This code will do the trick, and could be used to create the data set shown in Figure 7.25. However, this is an extra processing step. A better way to do this is to specify these changes in the ODS OUTPUT statement. When you ask for a data set in an ODS OUTPUT statement, all of the usual data set options apply. By adding parentheses after the SAS data set name, you can use options like KEEP=, RENAME=, and DROP=.

The following code accomplishes the same thing as the preceding combination of ODS and DATA step code. However, this code does it in a single pass of the data, saving resources.

```
      ODS OUTPUT "Parameter Estimates"=Params
  (KEEP=Parameter Estimate Probt RENAME=(Probt=Pvalue));
proc rsreg data=traffic;
  model fatal=mph;
run;
ODS OUTPUT CLOSE;
```

This code also produces the data set shown in Figure 7.25.

Figure 7.23

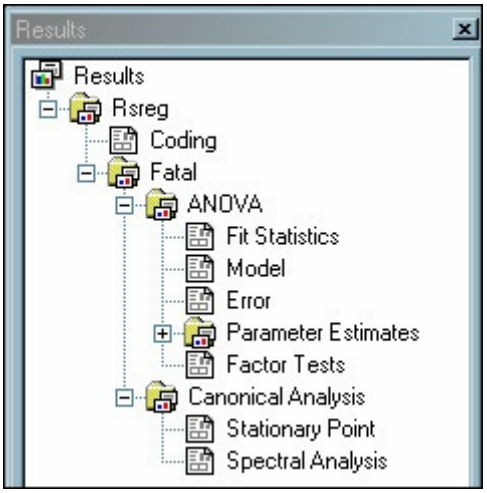


Figure 7.24

VIEWTABLE: Parameter Estimates

	Dependent	Parameter	DF	Estimate	StdErr	tValue	Probt	CodedEstimate
1	Fatal	Intercept	1	0.375440	0.096365	3.90	0.0001	0.248972
2	Fatal	MPH	1	-0.006994	0.007070	-0.99	0.3238	-0.025625
3	Fatal	MPH*MPH	1	0.000095473	0.000105	0.91	0.3638	0.100843

Figure 7.25

VIEWTABLE: Parameter Estimates

	Parameter	Estimate	Pvalue
1	Intercept	0.375440	0.0001
2	MPH	-0.006994	0.3238
3	MPH*MPH	0.000095473	0.3638

Creating ODS Output Data Sets from the PRINT and REPORT Procedures

For most procedures, you can generate at least one ODS output data set. A procedure where you cannot create an ODS output data set is PROC PRINT. Until SAS 9.2, you could not create an ODS output data set with PROC REPORT.

The reason PROC PRINT doesn't support ODS OUTPUT is that you are already printing a SAS data set, and so the output data set would match the input data set.

The PROC REPORT statement OUT= option enables you to create an output data set. You can also use the ODS OUTPUT statement to create an output data set with PROC REPORT. This ODS output data set is the same as the data set produced with the OUT= option in the PROC REPORT statement.

If you execute an ODS TRACE ON statement before either a PROC PRINT or PROC REPORT step, you will find that each produces an output object, as shown in Figure 7.26. However, if you submit an ODS OUTPUT statement to create a data set from the PRINT output object from PROC PRINT, ODS does not create an output data set and SAS writes an error message in the SAS log.

The following program produces the error message from PROC PRINT as shown in Figure 7.27. The subsequent code to create an ODS output data set with PROC REPORT works correctly and does not write an error message to the SAS log.

```
      ODS OUTPUT Print=PrintOut;
proc print data=traffic(obs=10);
run;
ODS OUTPUT CLOSE;
      ODS OUTPUT Report=RepOut;
      proc report data=traffic(obs=10) nowd;
        column highway location event impact mph timeofday;
      run;
ODS OUTPUT CLOSE;
```

Figure 7.26

```
94    ODS TRACE ON;
95
96    proc print data=traffic(obs=10);
97    run;


Output Added:
-------------
Name:        Print
Label:       Data Set WORK.TRAFFIC
Data Name:
Path:        Print.Print
-------------
NOTE: There were 10 observations read from the data set WORK.TRAFFIC.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


98    proc report data=traffic(obs=10) nowd;
99      column highway location event impact mph timeofday;
100   run;


Output Added:
-------------
Name:        Report
Label:
Data Name:   ProcReportTable
Path:        Report.Report.Report
-------------
NOTE: There were 10 observations read from the data set WORK.TRAFFIC.
NOTE: PROCEDURE REPORT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


101
102   ODS TRACE OFF;
```

Figure 7.27

```
107   ODS OUTPUT Print=PrintOut;
108   proc print data=traffic(obs=10);
109   run;

ERROR: PROC PRINT does not support the ODS OUTPUT statement.
NOTE: The SAS System stopped processing this step because of errors.
WARNING: Output 'Print' was not created.  Make sure that the output object name, label, or
         path is spelled correctly.  Also, verify that the appropriate procedure options are
         used to produce the requested output object.  For example, verify that the NOPRINT
         option is not used.
NOTE: There were 10 observations read from the data set WORK.TRAFFIC.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

110   ODS OUTPUT CLOSE;
```

Chapter 8
Limiting ODS Output

In general, when you run a SAS procedure and don't add ODS features, your output choices are pretty much all or nothing. Either you get pages of output, or if the procedure has the NOPRINT option, you get no output other than a data set.

When you add ODS features to your programs, you can have much more control over the amount of output your procedure steps produce and you can send components of the output to different destinations.

This chapter shows you how to use ODS to select or exclude specific output objects so that you can get exactly the output you want, in exactly the formats you want, for any procedure you want. For example, you can request a complete output data set, one table from the results in RTF, and another table from the results in CSV format. As long as your procedure produces output objects, you can select or exclude specific output objects.

This chapter uses the data set VOTERS to illustrate the concepts. This data set is available on the author pages for this book at http://support.sas.com/authors.

Selecting a Subset of Your Results

If you're using ODS to create output data sets, you have to specify each output object you want. With HTML, RTF, printer, and listing output, by default you get everything the procedure can produce. You can change what you get in limited ways by using certain procedure options. For example, the CHISQ option adds some results to PROC FREQ output, and the NOSIMPLE option removes some results from PROC CORR.

However, the best way to customize the content of your results is to use ODS. For example, if you have the following code that creates RTF output from some univariate statistics, you get the standard UNIVARIATE procedure output shown in Figure 8.1.

```
      ODS RTF FILE='Univariate.rtf';
proc univariate data=voters;
  var income;
run;
ODS RTF CLOSE;
```

This procedure produces a lot of output. So much so, in fact, that there isn't room to show all of the output on the facing page. If what we really want is just the basic statistics such as mean, median, and mode, we could save a lot of space by getting rid of the other information. One way to get these statistics would be to use ODS to create an output data set from the BasicMeasures output object, and then print the results to an RTF file.

A better way is to use the ODS SELECT statement. This statement enables you to list the names of the output objects you want to select. Here is the syntax:

ODS SELECT *output-object(s)-to-include*;

The objects you list are the only objects that are sent to your open output destinations. In this example, we can use ODS SELECT to request that the output be limited to the BasicMeasures object.

```
      ODS SELECT BasicMeasures;
ODS RTF FILE='UnivariateSelect.rtf';
proc univariate data=voters;
  var income;
run;
ODS RTF CLOSE;
```

Now the results easily fit into just a portion of the page, as shown in Figure 8.2.

You can use this technique with any output destination except ODS OUTPUT. This example used RTF output and the UNIVARIATE procedure, but the RTF specification could be traded for HTML or PDF, and the result would be the same. If listing output is turned on, then the selection applies to that destination too.

Figure 8.1: Partial Results

The UNIVARIATE Procedure
Variable: income (Income)

Moments			
N	276	Sum Weights	276
Mean	120433.087	Sum Observations	33239532
Std Deviation	60089.207	Variance	3610712795
Skewness	-0.0637537	Kurtosis	-1.2638647
Uncorrected SS	4.99609E12	Corrected SS	9.92946E11
Coeff Variation	49.8942678	Std Error Mean	3616.94522

Basic Statistical Measures			
Location		Variability	
Mean	120433.1	Std Deviation	60089
Median	124862.0	Variance	3610712795
Mode	.	Range	198416
		Interquartile Range	106549

Tests for Location: Mu0=0				
Test	Statistic		p Value	
Student's t	t	33.29691	Pr > |t|	<.0001

Figure 8.2

The UNIVARIATE Procedure
Variable: income (Income)

Basic Statistical Measures			
Location		Variability	
Mean	120433.1	Std Deviation	60089
Median	124862.0	Variance	3610712795
Mode	.	Range	198416
		Interquartile Range	106549

Selecting Multiple Output Objects

The SELECT statement is not limited to selecting just a single output object for display. You can pick and choose among all of the output objects that a procedure produces.

For example, if you've run a simple logistic model, you might look over the results and decide that you'd like to create a report that includes just part of the output. You find the AIC statistic useful, and also the parameter estimates, but you don't need things like the response profile or class level information.

By running an ODS TRACE, you discover that the AIC statistic is in the FitStatistics output object and the parameter estimates are in the ParameterEstimates object. So those two objects go in the ODS SELECT statement. The code for these selections is shown below, and the output is displayed in Figure 8.3.

One rule about placement of the ODS SELECT statement is that it must occur *before* the procedure finishes. So in this example, the ODS SELECT statement could appear after the ODS HTML statement, but it could *not* be placed after the RUN statement that ends the PROC LOGISTIC step. This form of the SELECT statement selects objects to send to all open destinations. There is a more selective form of the SELECT statement that we discuss in an upcoming topic.

```
      ODS SELECT FitStatistics ParameterEstimates;
ODS HTML BODY='SelectLogistic.html' STYLE=sasweb;
proc logistic data=voters;
  class gender party;
        model voted=gender party;
      run;
ODS HTML CLOSE;
```

That was easy, so let's look at another example. When you're exploring a data set for the first time, one of the first things you want to do is run a PROC CONTENTS to find out what's in the data set. The default output has a header with information such as the number of observations and variables. It also has a detailed listing of each variable. However, there is no option to view just the header information. If you have a data set with a lot of variables, the PROC CONTENTS output can be unwieldy.

ODS SELECT can solve this problem. If you run an ODS TRACE on a PROC CONTENTS step, you will discover that the header and the list of variables are two separate output objects. This means that you can select just the object for the header and not have to also list the variables.

So to get a PROC CONTENTS with just the header information, use ODS SELECT to request the Attributes object. The code is as follows, and the output is displayed in Figure 8.4.

```
      ODS SELECT Attributes;
ODS HTML BODY='SelectContents.html' STYLE=sasweb;
```

```
proc contents data=voters;
run;
ODS HTML CLOSE;
```

Figure 8.3

The LOGISTIC Procedure

Model Fit Statistics		
Criterion	Intercept Only	Intercept and Covariates
AIC	384.559	386.334
SC	388.180	400.816
-2 Log L	382.559	378.334

Analysis of Maximum Likelihood Estimates						
Parameter		DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept		1	0.0600	0.1365	0.1931	0.6604
gender	Female	1	0.2257	0.1216	3.4435	0.0635
party	Democrat	1	0.0295	0.1703	0.0301	0.8623
party	Independent	1	0.1443	0.2261	0.4075	0.5232

Figure 8.4

The CONTENTS Procedure

Data Set Name	WORK.VOTERS	Observations	276
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	Sat, Nov 15, 2008 02:58:50 PM	Observation Length	48
Last Modified	Sat, Nov 15, 2008 02:58:50 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Excluding the Output You Don't Want

If you're working with a procedure that produces a lot of output objects, you might find it easier to exclude the ones you don't want, rather than select all of the ones you do want. ODS provides an EXCLUDE statement in addition to the SELECT statement. The syntax is exactly the same:

ODS EXCLUDE *output-object(s)-to-exclude*;

To compare the two statements, let's look at a sample program. First, we'll identify our chosen output objects using the ODS TRACE statement, then we'll use the ODS SELECT statement, and last we'll repeat the program using the EXCLUDE statement.

For this example, we modify the output from the UNIVARIATE procedure to eliminate the Extreme Observations table. To code this using the SELECT statement, we need to name every output object that we want except for the ExtremeObs object. Here is the code:

```
      ODS SELECT Moments BasicMeasures TestsForLocation Quantiles;
ODS PDF FILE='SelectUniv.pdf';
proc univariate data=voters;
   var income;
run;
ODS PDF CLOSE;
```

Because PROC UNIVARIATE produces five output objects, to exclude one you need to select the other four. This code produces the output objects listed in the SAS Results window in Figure 8.5.

We can produce exactly the same output using the EXCLUDE statement. With EXCLUDE, all we have to do is list the one object we want to exclude: the ExtremeObs object. The code is shown below.

```
      ODS EXCLUDE ExtremeObs;
ODS PDF FILE='SelectUniv2.pdf';
proc univariate data=voters;
   var income;
run;
ODS PDF CLOSE;
```

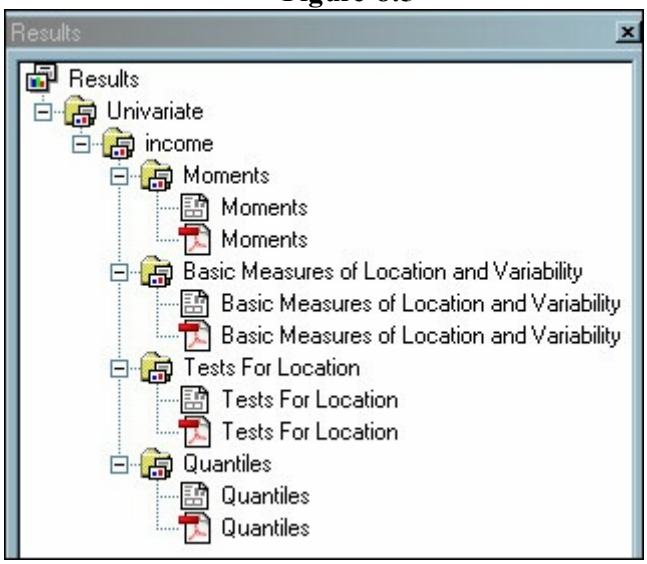
This code also produces the Results window output shown in Figure 8.5. The output files from these two versions of the code are identical.

It's up to you whether you use SELECT or EXCLUDE. You can achieve exactly the same results with either statement. Generally, you want to use the statement that involves the least typing. So if you have more objects to keep than to remove, use EXCLUDE. On the other hand, if you have more objects to get rid of than keep, use SELECT.

Another consideration is documentation. If you want your code to be self-documenting, you might find it helpful to have a list of the selected output objects hard-coded into your

program. SELECT does this for you, whereas EXCLUDE shows only what was left out.

Figure 8.5



Avoid using ODS SELECT and ODS EXCLUDE in the same program. These two statements can interact and generate unexpected results.

Selecting Output for Multiple Procedures

The ability to limit your output is especially handy when you're doing repeated runs of the same procedure. For example, you might be running a series of models or a series of tables. If you were forced to keep all of the output from all of the procedures, your results could get unwieldy.

Using SELECT or EXCLUDE enables you to build a concise summary report from your results. For example, let's say you are running a bunch of crosstabulations, trying to identify variables with significant relationships. You can do this with a series of FREQ procedure steps. Because you only want to know which variables are significantly related, you can save space in the output by just reporting the statistics, not the actual crosstabulations.

The following code uses the SELECT statement to request only the ChiSq output object. It has two PROC FREQ steps that are used to create an RTF report. The STARTPAGE=NO option in the ODS RTF statement puts both chi-square tables on the same page.

```
      ODS SELECT ChiSq;
ODS RTF FILE='SelectFreq.rtf' STARTPAGE=NO;
proc freq data=voters;
  table voted*party / chisq;
run;
proc freq data=voters;
  table voted*gender / chisq;
run;
ODS RTF CLOSE;
```

However, we need to make a couple of additions before running this code. To use SELECT or EXCLUDE with multiple procedures, you need to add a new option. Following the output object, you need to add the PERSIST option. The syntax calls for it to be placed in parentheses after each output object.

Another change comes at the end of the code. In order to keep this selection from continuing until the end of the SAS session, it's best to follow your procedures with an ODS SELECT ALL statement to reset the select list. If you've used ODS EXCLUDE with the PERSIST option, add an ODS EXCLUDE NONE statement at the end to reset the selections.

```
      ODS SELECT ChiSq (PERSIST);
ODS RTF FILE='SelectFreq.rtf' STARTPAGE=NO;
proc freq data=voters;
  table voted*party / chisq;
run;
proc freq data=voters;
  table voted*gender / chisq;
run;
ODS RTF CLOSE;
ODS SELECT ALL;
```

If you look at the Results window shown in Figure 8.6, you can see that only the ChiSq output object has been created by each procedure. The resulting RTF file is shown in Figure 8.7. The two ChiSq objects appear together on one page because of the STARTPAGE=NO

option.

Figure 8.6

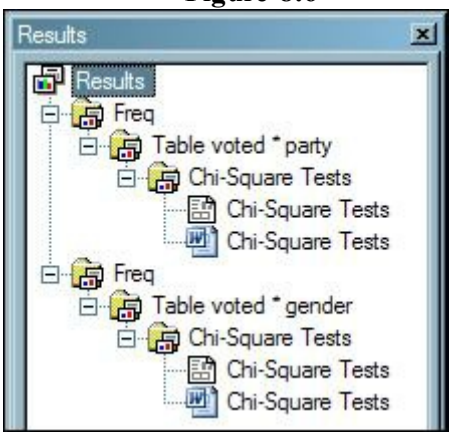


Figure 8.7

The FREQ Procedure

Statistics for Table of voted by party

Statistic	DF	Value	Prob
Chi-Square	2	0.7567	0.6850
Likelihood Ratio Chi-Square	2	0.7573	0.6848
Mantel-Haenszel Chi-Square	1	0.3968	0.5287
Phi Coefficient		0.0524	
Contingency Coefficient		0.0523	
Cramer's V		0.0524	

The FREQ Procedure

Statistics for Table of voted by gender

Statistic	DF	Value	Prob
Chi-Square	1	3.2680	0.0706
Likelihood Ratio Chi-Square	1	3.2746	0.0704
Continuity Adj. Chi-Square	1	2.8472	0.0915
Mantel-Haenszel Chi-Square	1	3.2562	0.0712
Phi Coefficient		0.1088	
Contingency Coefficient		0.1082	
Cramer's V		0.1088	

Customizing Your Selections for Each Output Destination

Using the SELECT or EXCLUDE statements to limit your results can be helpful, but you need to be careful. By dropping some of the output, you might miss something important. For example, what if the frequency table in the previous example contained some unexpected missing data? You might not spot the problem by just looking at the ChiSq output object.

The best way to avoid this problem is to produce your desired output object reports using HTML, RTF, or PDF output; but keep the full output in the Output window. This way you can review the detailed listing output before you start using the summary report.

Both the SELECT and EXCLUDE statements can be applied to a specific output destination. All the previous examples have not specified a destination, causing the statements to apply to all of the open destinations. If you want the selections or exclusions to apply to only one output destination, you just need to add the appropriate destination. For example, to exclude an object from the RTF destination, the syntax is as follows:

ODS RTF EXCLUDE *output-object-name*;

Alternatively, to select an object for the HTML destination, you use the following syntax:

ODS HTML SELECT *output-object-name*;

As an example, we can create a summary report with just the correlation coefficients from PROC CORR and keep the full listing output separate for detailed review. This enables us to review the basic statistics shown in the listing to be sure that the data is what we expected. Here is the code:

```
ODS HTML BODY='SelectCorr.html' STYLE=SASWEB;
ODS HTML SELECT PearsonCorr;
proc corr data=voters;
  var income;
  with age;
run;
ODS HTML CLOSE;
```

The SELECT statement must come after the HTML destination has been opened. When you issue a general SELECT statement that does not apply to any one destination, you can list it before or after a destination is opened, as long as it comes before the statements that generate the results you want to capture. When you specify a single output destination, that destination must be open in order for the SELECT or EXCLUDE to work. Otherwise, you will get a warning message in your SAS log, and the SELECT or EXCLUDE will have no effect.

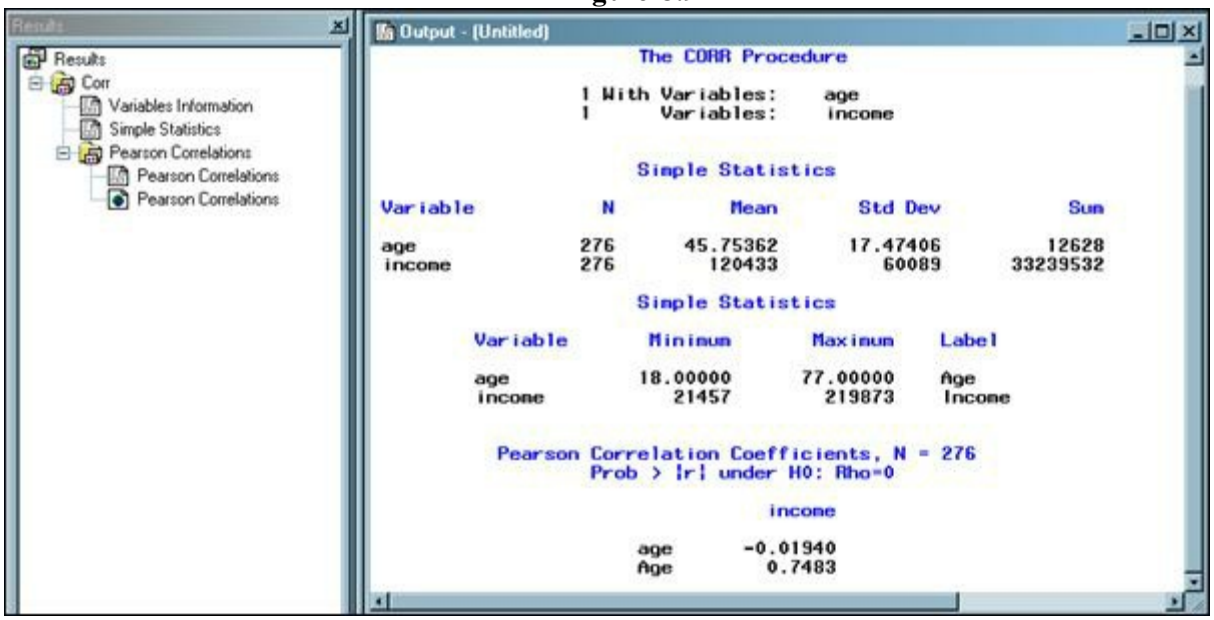
The HTML file created by this code is shown in Figure 8.8. Notice how it contains only the correlation coefficients. The listing output contains the full results, and is shown in Figure 8.9. By using SELECT or EXCLUDE with a specific output destination, you get the best of both worlds: a short summary report and a detailed listing for review.

Figure 8.8

The CORR Procedure

Pearson Correlation Coefficients, N = 276 Prob > |r| under H0: Rho=0	
	income
age	-0.01940
Age	0.7483

Figure 8.9



Selecting Complex Combinations of Output Objects

In Chapter 6, “Exploring ODS Output,” you saw a number of examples of identifying output objects by their labels. In this chapter, you have seen a number of examples of identifying output objects by their names. These are the simplest ways to identify an output object when you are creating an output data set or using the ODS SELECT or ODS EXCLUDE statements. However, these are not the only ways.

The following code uses a number of approaches to identify the output objects in its ODS SELECT statements. To see how these methods relate to the output objects, you need to review the ODS TRACE output shown in Figure 8.10. This is not a complete trace of the code, but rather a selection of tracings, one for each type of output object.

```
      ODS LISTING CLOSE;
ODS HTML BODY='SelectMany.htm' STYLE=SASWEB;
ODS HTML SELECT Freq.Table1.OneWayFreqs
               Table2.OneWayFreqs
               "Table party"."One-Way Frequencies"
               "One-Way Chi-Square Test";
proc freq data=voters;
   tables gender voted party / chisq;
run;
ODS HTML SELECT ExtremeObs#1 Quantiles#2;
proc univariate data=voters;
   var income age;
run;
ODS HTML CLOSE;
```

First let’s look at the selections for the PROC FREQ step and the trace information in Figure 8.10. The first selection, Freq.Table1.OneWayFreqs, is an example of selecting an object by its path. The second part of the name, “Table1,” refers to the analysis for the variable GENDER.

You can also identify an object by a partial path, as in the second selection, Table2.OneWayFreqs. When using a partial path, you can start with any portion of the path, but then you have to include everything to the right of where you start. So OneWayFreqs is a legal path, but Freq.Table2 would not be correct. The first part of the name, “Table2,” refers to the analysis for the variable VOTED.

The next two selections demonstrate the use of the label path. You can see the label path in Figure 8.10, but you can also find this path by looking at the Results window. If you combine the labels at each level of the results in the Results window, you get the label path. The third and fourth selections in the code above demonstrate the use of partial label paths (you can also use a full label path, but those can get pretty long).

The net effect of the four selections before the PROC FREQ is to select only the output objects that are desired. The three frequency tables are picked individually by the first three selections, and the three chi-square tables are picked by the fourth selection.

The second SELECT statement applies to the PROC UNIVARIATE step. It shows one

more way to identify output. If you have an output object that will be created more than once, you can identify each occurrence by number. ExtremeObs#1 is the extreme observations table for the variable INCOME, and Quantiles#2 is the quantiles table for the variable AGE.

To see how all of these selections work, the Results window from this run is shown in Figure 8.11.

Figure 8.10: Sample Output Tracings

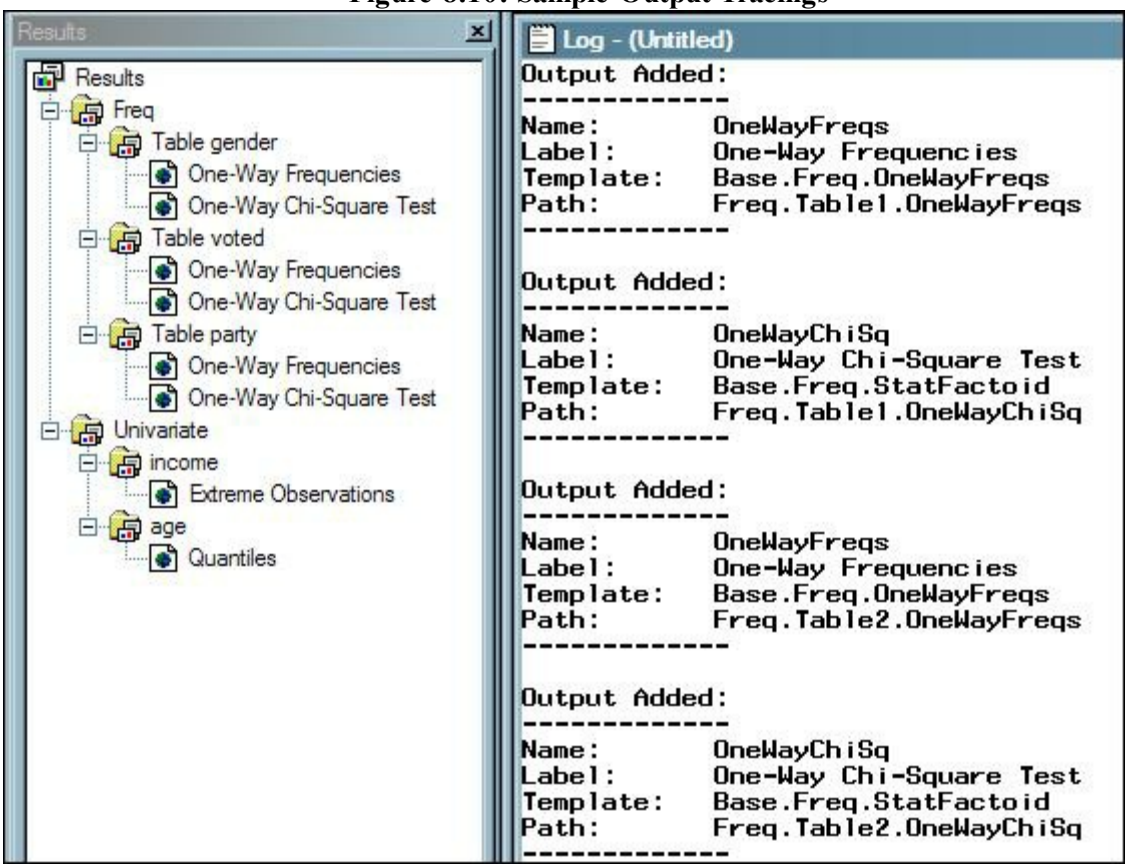
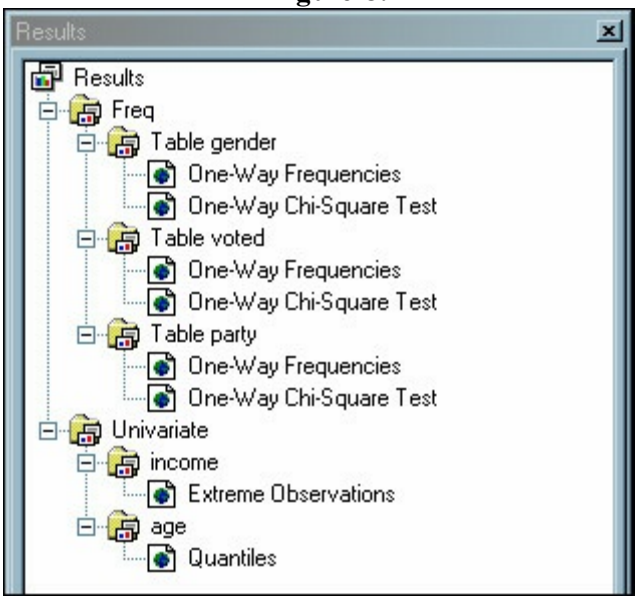


Figure 8.11



Identifying the Current Selections and Exclusions

The examples shown so far have been simple. You could easily see what was being

selected and excluded. However, as you start to create complex production jobs with thousands of lines of code, you might start running into problems. The following example shows how to debug a problem with your ODS output.

This code should create an RTF file with two chi-square results tables. The EXCLUDE statement blocks the display of the crosstabulations. However, if you view Figure 8.12, you can see that the resulting RTF file is empty.

```
      ODS RTF FILE='selectshow.rtf';
ODS RTF EXCLUDE CrossTabFreqs(Persist);
proc freq data=voters;
  tables party*gender / chisq;
run;
proc freq data=voters;
  tables party*voted / chisq;
run;
ODS RTF CLOSE;
```

What happened to the results? The first thing to do is check the log for errors. In this case, the log shows no error messages. It indicates that the requested RTF file was created. The next thing to do is look at the listing output. This output is not shown here, but it contains only crosstabulations and not the chi-square results.

So, what is going wrong? These lines of code are actually part of a larger program, and at this point you should be suspicious of the preceding code. The SELECT and EXCLUDE statements are powerful tools, but if you are not careful, they can keep you from getting the desired results. To find out if the problem is with the SELECT or EXCLUDE statements, we need to use a new ODS statement, the SHOW statement. This allows you to see the current selection and exclusion lists. The statement is added after the other ODS RTF statements and right before the procedures are called.

```
      ODS RTF SHOW;
```

The ODS RTF SHOW statement sends its results to the SAS log, as shown in Figure 8.13. According to these results, our exclusion list is exactly as expected. The only item being excluded is the CrossTabFreqs object. So this does not solve our problem. The solution comes from adding a second ODS SHOW statement. The ODS RTF SHOW statement lists the current selection and exclusion lists for the RTF destination. However, it is also possible to select and include objects across all destinations. So to see what is going on, we need to add a SHOW statement for all destinations.

```
      ODS SHOW;
ODS RTF SHOW;
```

Now we have our answer. Figure 8.14 contains the lines written to the log file, and you can see that our problem is that we have an overall exclusion in effect in addition to our exclusion for the RTF destination. The overall exclusion removed the ChiSq output object from all destinations. What's going on is that somewhere in the preceding code, we used an ODS EXCLUDE with a PERSIST option, and forgot to clear the exclusion with a subsequent ODS EXCLUDE NONE statement. This allowed the earlier code to affect output later in the

program.

Figure 8.12

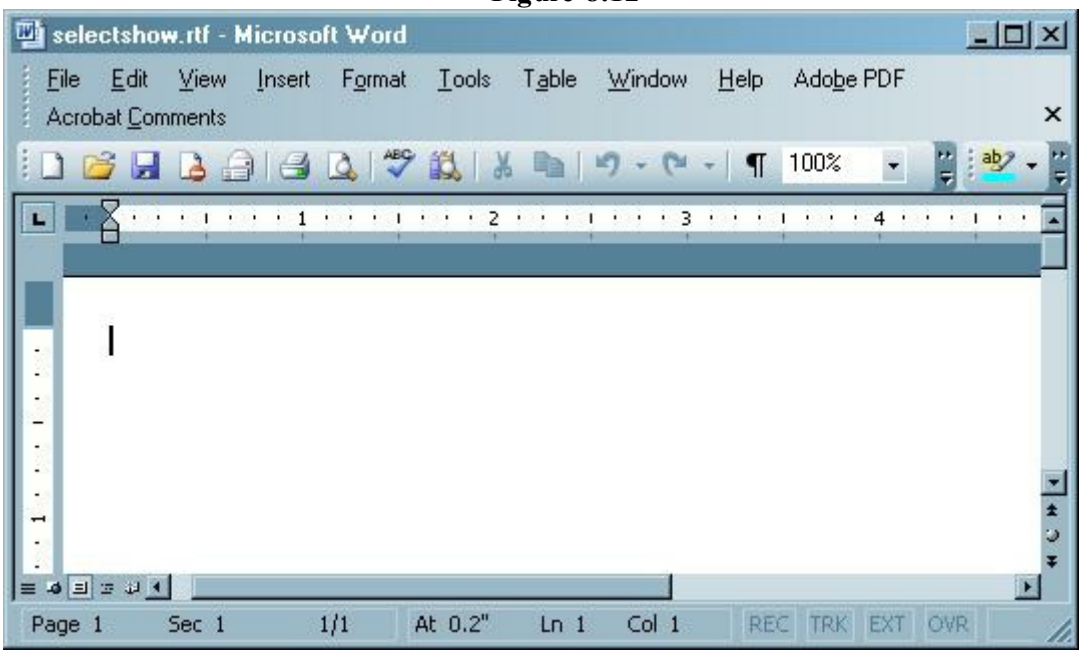


Figure 8.13

```
306  ODS RTF SHOW;
Current RTF exclude list is:
1. CrossTabFreqs(PERSIST)
```

Figure 8.14

```
331  ODS SHOW;
Current OVERALL exclude list is:
1. CHISQ(PERSIST)
332  ODS RTF SHOW;
Current RTF exclude list is:
1. CrossTabFreqs(PERSIST)
```

If you are using the PERSIST option, be careful to follow it with an ODS SELECT ALL or ODS EXCLUDE NONE statement to keep it from affecting other parts of your program. If you do get into trouble, remember to use the ODS SHOW statement.

Selecting with WHERE Conditions

Starting in SAS 9.2, you have a more advanced ability to select (and exclude) output objects. The examples shown so far have been simple. What if, for some reason, you wanted all the output objects related to the INCOME variable from a job to go to the RTF destination and all the output objects related to the AGE variable to go to the HTML destination?

It would be tedious to run your procedures and then hard code ODS SELECT (or EXCLUDE) statements for all the output objects of interest. However, an ODS TRACE statement reveals that variable names appear in either the path or the label path information from the ODS TRACE.

Armed with the knowledge that the SELECT and EXCLUDE statements support WHERE processing, we could code the following program with WHERE= conditions in the SELECT statements:

```
      ODS RTF FILE='SelectIncome.rtf';
ODS HTML BODY='SelectAge.html' STYLE=SASWEB;
ODS RTF SELECT WHERE=(upcase(_path_) contains 'INCOME' or
                 upcase(_labelpath_) contains 'INCOME');
ODS HTML SELECT WHERE=(upcase(_path_) ? 'AGE' or
                  upcase(_labelpath_) ? 'AGE');
proc univariate data=voters;
  var income age;
run;
proc freq data=voters;
  tables age / chisq;
run;
ODS HTML CLOSE;
ODS RTF CLOSE;
```

All we needed to do to code the correct WHERE= condition was review the ODS TRACE output to verify that our variable name of interest appeared in either the path or label path information, as shown by the partial trace information in Figure 8.15. Then, if we look at the RTF and HTML files, we see that each file contains information for only the variables of interest, as shown in Figures 8.16 and 8.17 respectively.

Of course, this was a somewhat artificial example, but the ability to use WHERE logic in your ODS SELECT or EXCLUDE statements makes this powerful feature even more powerful. The keywords that you can use in your WHERE= condition include the _PATH_, _LABELPATH_, _NAME_, and _LABEL_. Regular WHERE= condition operators are supported.

Figure 8.15

Partial TRACE Results

```
Output Added:
-------------
Name:        Moments
Label:       Moments
Template:    base.univariate.Moments
Path:        Univariate.income.Moments
Label Path:  'The Univariate Procedure'.'income'.'Moments'
-------------

Output Added:
-------------
Name:        Moments
Label:       Moments
Template:    base.univariate.Moments
Path:        Univariate.age.Moments
Label Path:  'The Univariate Procedure'.'age'.'Moments'
-------------

Output Added:
-------------
Name:        OneWayFreqs
Label:       One-Way Frequencies
Template:    Base.Freq.OneWayFreqs
Path:        Freq.Table1.OneWayFreqs
Label Path:  'The Freq Procedure'.'Table age'.'One-Way Frequencies'
-------------
```

```
Output Added:
-------------
Name:        OneWayChiSq
Label:       One-Way Chi-Square Test
Template:    Base.Freq.StatFactoid
Path:        Freq.Table1.OneWayChiSq
Label Path:  'The Freq Procedure'.'Table age'.'One-Way Chi-Square Test'
-------------
```

Figure 8.16: Partial RTF Output for INCOME

The UNIVARIATE Procedure
Variable: income (Income)

Moments			
N	276	Sum Weights	276
Mean	120433.087	Sum Observations	33239532
Std Deviation	60089.207	Variance	3610712795
Skewness	-0.0637537	Kurtosis	-1.2638647
Uncorrected SS	4.99609E12	Corrected SS	9.92946E11
Coeff Variation	49.8942678	Std Error Mean	3616.94522

Basic Statistical Measures			
Location		Variability	
Mean	120433.1	Std Deviation	60089
Median	124862.0	Variance	3610712795
Mode	.	Range	198416
		Interquartile Range	106549

Figure 8.17: Partial HTML Output for AGE

The UNIVARIATE Procedure
Variable: age (Age)

Moments			
N	276	Sum Weights	276
Mean	45.7536232	Sum Observations	12628
Std Deviation	17.4740583	Variance	305.342714
Skewness	0.0520333	Kurtosis	-1.2372552
Uncorrected SS	661746	Corrected SS	83969.2464
Coeff Variation	38.1916384	Std Error Mean	1.05181471

The FREQ Procedure

Age				
age	Frequency	Percent	Cumulative Frequency	Cumulative Percent
18	9	3.26	9	3.26
19	4	1.45	13	4.71
20	7	2.54	20	7.25
21	3	1.09	23	8.33
22	6	2.17	29	10.51
23	9	3.26	38	13.77

Chapter 9
Using ODS in the DATA Step

This chapter presents examples of DATA step programs that produce reports using ODS. It describes the ODS features that you can use in DATA step programs to structure and customize the report output. DATA step reports in tabular form can be routed to any ODS destination such as RTF, PDF, or HTML by using the same ODS "sandwich" statements that you used with SAS procedures in previous examples.

DATA step functionality for creating, manipulating, and transforming data sets or creating external data files is outside the scope of this book. There are other books that investigate data manipulation and transformation using DATA step programs. However, one special subset of DATA step syntax harnesses ODS capability and enables you to create highly customized tabular reports.

If you normally create output data sets with the DATA step and then immediately follow your program with a PROC PRINT step, this chapter shows you how you can add some very simple statements to your DATA step program in order to perform your data manipulation and create your tabular report in one step.

In addition, you can use DATA step logic to control what is written out to each column of your tabular report. And you can use FIRST.BYVAR and LAST.BYVAR logic to control whether you write summary lines to the final report, based on BY groups.

This chapter uses the data set TRAFFIC to illustrate the examples. This data set is available on the author pages for this book at http://support.sas.com/authors.

Creating ODS Output in the DATA Step Using FILE and PUT Statements

When you use the DATA step syntax designed for ODS, you can produce tabular output in the default proportional font for the destination. The following DATA step sends a report to the three named destinations of RTF, PDF, and HTML. Assuming that the LISTING destination is open, the DATA step also sends the report to the LISTING destination.

```
ODS RTF FILE='BasicReport.rtf';
ODS PDF FILE='BasicReport.pdf';
ODS HTML FILE='BasicReport.html' STYLE=sasweb;
title 'Default PUT _ODS_ Output of All Variables';
data _null_;
  set traffic;
  file print ods;
  put _ods_;
RUN;
ODS _ALL_ CLOSE;
```

The only two statements that need modification to send output to an ODS destination are the FILE PRINT and PUT statements. The ODS specification in the FILE PRINT statement requests that ODS format the output produced by the DATA step. The _ODS_ specification in the PUT statement causes the DATA step to list in the ODS destination all the variables in the Program Data Vector (PDV). The DATA step lists the variables in the order of their placement in the PDV. Note that column pointers are absent from the PUT statement. Figures 9.1, 9.2, 9.3, and 9.4 show the output from the program for the RTF, the PDF, the HTML, and the LISTING destinations respectively.

The important thing to note about the output is that it is basically tabular in form. The column names or labels represent the headers for the table, and the data values for each column appear underneath their respective header.

Although this report closely resembles PROC PRINT output, we have produced it from within a DATA step program.

At this time, it is not possible to create completely free-format DATA step reports when sending them to a destination other than LISTING. The reason is that ODS cannot process exact column locations when formatting output for a destination that's using a proportional font. The widths of characters vary in a proportional font like Arial, whereas they are all the same in monospace fonts like Courier. Because the LISTING destination uses monospace fonts, you can specify exact column locations when using the classic DATA step report-writing technique. In effect, with this classic technique, you treat your output file as though it were a big piece of graph paper, and every cell on the graph paper (or output page) is addressable with the PUT statement.

ODS developers are working on an interface between the DATA step and ODS to allow the creation of free-format output. For more information about this capability and for examples of classic report-writing programs and output, refer to the SAS Web site.

Figure 9.1

Default PUT _ODS_ of All Variables

Highway	Location	Event	Effect on Traffic	Average Speed	Time (24 hour format)	Accident Fatal
280	San Francisco	Injury	Major Jam	0	15	0
80	San Francisco	Injury	Major Jam	18.5	8	0
101	South Bay	Disabled	Major Jam	5.5	17	0
80	South Bay	Disabled	Slowdown	31	9	0

Figure 9.2

Default PUT _ODS_ of All Variables

Highway	Location	Event	Effect on Traffic	Average Speed	Time (24 hour format)	Accident Fatal
280	San Francisco	Injury	Major Jam	0	15	0
80	San Francisco	Injury	Major Jam	18.5	8	0
101	South Bay	Disabled	Major Jam	5.5	17	0
80	South Bay	Disabled	Slowdown	31	9	0

Figure 9.3

Default PUT _ODS_ of All Variables

Highway	Location	Event	Effect on Traffic	Average Speed	Time (24 hour format)	Accident Fatal
280	San Francisco	Injury	Major Jam	0	15	0
80	San Francisco	Injury	Major Jam	18.5	8	0
101	South Bay	Disabled	Major Jam	5.5	17	0
80	South Bay	Disabled	Slowdown	31	9	0

Figure 9.4

Default PUT _ODS_ of All Variables 1

Highway	Location	Event	Effect on Traffic	Average Speed	Time (24 hour format)	Accident Fatal
280	San Francisco	Injury	Major Jam	0	15	0
80	San Francisco	Injury	Major Jam	18.5	8	0
101	South Bay	Disabled	Major Jam	5.5	17	0
80	South Bay	Disabled	Slowdown	31	9	0

Naming and Labeling the DATA Step Output Object

The previous chapters showed some examples that changed the top level node name in the Results window by including the ODS PROCLABEL statement. When you use ODS in the DATA step, the ODS PROCLABEL statement does not change the name of the top level node. However, you can change its name in a DATA step that sends output to an ODS destination by adding the OBJECT= and OBJECTLABEL= suboptions to the ODS option in the FILE PRINT statement. The two suboptions are added to the FILE PRINT statement in the following program.

```
ODS RTF FILE='LabelODS.rtf';
ODS PDF FILE='LabelODS.pdf';
ODS HTML FILE='LabelODS.html' STYLE=sasweb;
title 'Default PUT _ODS_ of All Variables';
data _null_;
  set traffic;
  FILE PRINT ODS=(OBJECT=objlink OBJECTLABEL='My Label');
put _ods_;
RUN;
ODS _ALL_ CLOSE;
```

The program produces reports identical to those shown in Figures 9.1, 9.2, and 9.3. The SAS Results window shown in Figure 9.5 reveals that the navigation links to each output file are labeled with the text specified to the OBJECTLABEL= suboption. A table of contents also uses these labels. If you want to see where the object name itself is used, right-click an object and select **Properties** to open the Properties window. The HTML object properties, which are shown in Figure 9.6, illustrate how the OBJECT= suboption is used to build the output object name.

By default, the DATA step interface to ODS names objects by attaching serial numbers to the object names, similar to the naming convention shown for procedure objects. Figure 9.6 shows that the HTML results are named Datastep#1.objlink#1.**objlink#3.** ODS names the PDF results Datastep#1.objlink#1.**objlink#2**,and it names the RTF results Datastep#1.objlink#1.**objlink#1.**

The number assigned to the last item in the list for an output (the bold items above) reflects the processing order in the ODS invocation; the HTML invocation for this single DATA step program was the third in the list of ODS invocation statements. If we added another DATA step program to the preceding example, you would see that the object names for the second DATA step would start with Datastep#2. Figure 9.7 shows the Properties window for the RTF output.

```
ODS RTF FILE='LabelODS.rtf';
ODS PDF FILE='LabelODS.pdf';
ODS HTML FILE='LabelODS.html' STYLE=sasweb;
title 'Default PUT _ODS_ of All Variables';
data _null_;
  set traffic;
  FILE PRINT ODS=(OBJECT=other OBJECTLABEL='Other Label');
put _ods_;
```

```
RUN;
ODS _ALL_ CLOSE;
```

Figure 9.5

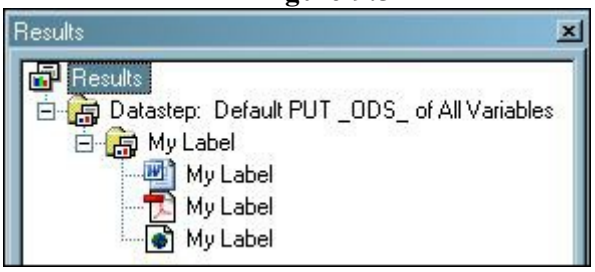


Figure 9.6

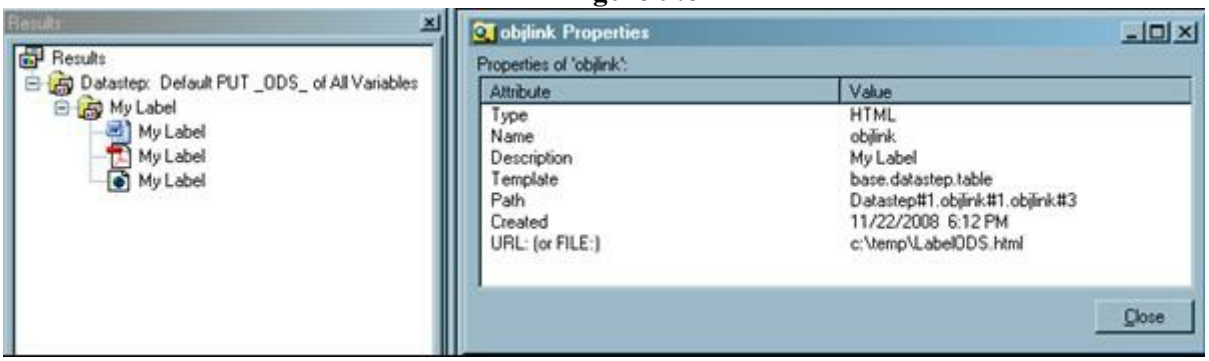
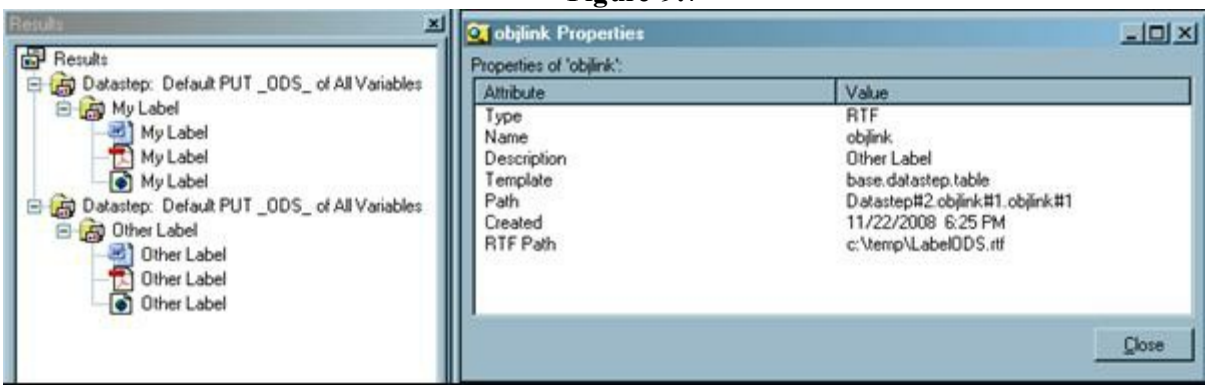


Figure 9.7



Selecting Variables to Include in the Output

The default action of the PUT _ODS_ statement is to list in tabular form all the variables in the Program Data Vector (PDV). This example shows you two ways to select the variables that you want the PUT _ODS_ statement to list.

The standard DROP/KEEP statement and data set option methods allow you to select the variables to include in the output, but they do not allow you to control the order in which the variables appear. The KEEP= data set option in the SET statement in the following DATA step specifies five of the seven variables in data set TRAFFIC to list in the output. Figure 9.8 shows the HTML results.

```
ODS HTML file='keepvar.html' STYLE=sasweb;
ODS PDF file='keepvar.pdf';
title 'Using the KEEP= Data Set Option';
data _null_;
  set traffic(keep=location event highway timeofday impact);
  file print ods;
  put _ods_;
run;
ODS _ALL_ CLOSE;
```

Although the KEEP= data set option specifies the variables to list in the output, it does not control the order in which to list them. The results in Figure 9.8 show that ODS lists the five variables in the order they appear in the PDV, and this is not the same order as they appear in the KEEP= option.

To include variables in the output and control the order of their appearance in the output, you need to use the VARIABLES= suboption of the ODS= option in the FILE PRINT statement. The set of variables to list and the order in which to list them is enclosed in parentheses.

```
ODS HTML file='varopt.html' STYLE=sasweb;
ODS PDF file='varopt.pdf';
title 'Using the VARIABLES= Suboption on the ODS= Option';
data _null_;
  set traffic;
  file print ods=(variables=(location event highway
                             timeofday impact));
  put _ods_;
run;
ODS _ALL_ CLOSE;
```

Figure 9.9 presents the PDF output from this program.

Figure 9.8

Using the KEEP= Data Set Option

Highway	Location	Event	Effect on Traffic	Time (24 hour format)
280	San Francisco	Injury	Major Jam	15
80	San Francisco	Injury	Major Jam	8
101	South Bay	Disabled	Major Jam	17
80	South Bay	Disabled	Slowdown	9

Figure 9.9

Using the VARIABLES= Suboption on the ODS= Option

Location	Event	Highway	Time (24 hour format)	Effect on Traffic
San Francisco	Injury	280	15	Major Jam
San Francisco	Injury	80	8	Major Jam
South Bay	Disabled	101	17	Major Jam
South Bay	Disabled	80	9	Slowdown

Controlling the Layout of the Output

This example shows you ways of using statements other than PUT _ODS_ to create ODS output in the DATA step. These additional ODS features can control the layout and appearance of the output through the use of column pointer control and conditional SAS language programming logic.

The meaning of "column pointer control" with ODS and the DATA step differs from the classic report-writing DATA step where SAS writes a value to a particular print position. With ODS and the DATA step, "column pointer control" refers to a tabular column in the output and not the literal character position in the row. For example, consider the following FILE PRINT ODS statement written in the style we saw in the previous example.

```
file print ods=(variables=(impact event location highway ));
```

Another way of requesting four variables in a specific order is to use the PUT statement with ODS column pointer control as shown below. This statement puts IMPACT variable values in column 1, EVENT variable values in column 2, LOCATION variable values in column 3, and HIGHWAY variable values in column 4.

```
put @1 impact @2 event @3 location @4 highway;
```

Note the following three restrictions in using FILE PRINT ODS:

The PDV must contain variables.

You cannot request ODS to write values in a column greater than the number of variables in the PDV or in a column greater than the number of variables in the VARIABLES= suboption. For example, you cannot write a value to column 5 if there are not at least five variables in the PDV or five variables in the VARIABLES= suboption.

You cannot write a text string to a column that belongs to or is used for a numeric variable.

Let's start with an example that writes values to specific columns and includes BY-group processing. Assume SAS sorted the data set TRAFFIC before the DATA step.

```
ODS HTML file='collayout.html' STYLE=sasweb;
title 'Controlling Layout';
data _null_;
  set traffic;
  by impact;
  file print ods=(variables=(impact event location highway ));
  if first.impact then do;
     put _ods_;
  end;
  else do;
    put @2 event @3 location @4 highway;
  end;
  if last.impact then put @1 ' ';
run;
ODS HTML CLOSE;
```

When processing the first observation in each BY group, the DATA step writes to the ODS HTML destination the values of all four variables. For all other observations within a BY group, the DATA step does not write the current value of IMPACT and lists only the values for EVENT, LOCATION, and HIGHWAY in columns 2, 3, and 4 respectively. When processing the last observation in the BY group, the DATA step writes a blank in column one so that there will be a blank line between BY groups in the report. The output for the first few observations and the last few observations in the first BY group is shown in Figure 9.10.

Figure 9.10

Controlling Layout

Effect on Traffic	Event	Location	Highway
Major Jam	Injury	San Francisco	280
	Injury	San Francisco	80
	Disabled	South Bay	101
	Disabled	San Francisco	101
	Disabled	San Francisco	80
	Minor	San Francisco	280
	Minor	San Francisco	280
	Minor	South Bay	80

. . .

	Disabled	San Francisco	101
	Injury	San Francisco	280
	Minor	San Francisco	101
	Disabled	San Francisco	280
Slowdown	Disabled	South Bay	80

Labeling and Formatting the Columns of the Results

If the variable you write with a DATA step to your ODS destination has a label, ODS uses the label as the variable's column heading. Similarly, if a format has been assigned to a variable, ODS writes the formatted values in the column. Variables in the data set TRAFFIC are labeled, and the previous example uses the variable labels for the four variables as column headers.

The previous behavior also holds true if you include ATTRIB, LABEL, or FORMAT statements in the same DATA step that creates the ODS report. The following DATA step uses an ATTRIB statement to override existing labels for two variables in the report and to apply format $HWYRT to the values of variable HIGHWAY. Format $HWYRT puts "I-" (for Interstate) or "US" (for US route) in front of the highway name.

```
ODS HTML FILE='FormatsLabels.html' STYLE=sasweb;
title 'DATA Step that Modifies Formats and Labels';
proc format;
  value $hwyrt '80'='I-80'
               '101'='US 101'
               '280'='I-280';
run;
data _null_;
   set traffic;
   file print ods=(variables=(impact event location highway));
   put _ods_;
   attrib impact label = 'Accident Impact'
          highway label = 'Hwy' format=$hwyrt.;
run;
ODS HTML CLOSE;
```

A second way of specifying labels and formats within the DATA step that produces the ODS report is to use the VARIABLES= suboption in the ODS= option of the FILE PRINT ODS statement, as shown in the following DATA step.

```
ODS HTML FILE="FormatsLabels.html" STYLE=sasweb;
title 'DATA Step that Modifies Formats and Labels';
data _null_;
   set traffic;
   file print ods=(variables=(
            impact(label="Accident Impact")
            event
            location
            highway(format=$hwyrt. label="Hwy")
            ));
   put _ods_;
run;
ODS HTML CLOSE;
```

Note the multiple levels of parentheses required with this syntax. It might be easier to check whether the parentheses are matched if you list each variable on a separate line as shown in the code.

The output from both DATA steps is identical and is shown once in Figure 9.11.

Figure 9.11

DATA Step that Modifies Formats and Labels

Accident Impact	Event	Location	Hwy
Major Jam	Injury	San Francisco	I-280
Major Jam	Injury	San Francisco	I-80
Major Jam	Disabled	South Bay	US 101
Major Jam	Disabled	San Francisco	US 101
Major Jam	Disabled	San Francisco	I-80
Major Jam	Minor	San Francisco	I-280
Major Jam	Minor	San Francisco	I-280
Major Jam	Minor	South Bay	I-80
Major Jam	Fatal	San Francisco	I-280
Major Jam	Fatal	San Francisco	I-80
Major Jam	Fatal	San Francisco	US 101
Major Jam	Fatal	San Francisco	I-280
Major Jam	Minor	San Francisco	US 101

Modifying Output with the PUT Statement

The example “Controlling the Layout of the Output” earlier in this chapter placed blank lines between BY groups of a report. Let’s refine that example so that when the DATA step processes the last observation in a BY group, it lists informative text and the total number of events for that BY group, rather than just writing a blank line.

The program uses standard DATA step logic to retain the total variable ITOT, which will contain the number of events in a BY group. When processing the first observation in a BY group, the DATA step initializes its value to 1. Its value is incremented by 1 in the ELSE DO group each time the DATA step processes a subsequent observation in the BY group. The last PUT statement executes when the DATA step processes the last observation in a BY group. This statement writes the text “Tot Event” in column 1 and the character representation of the numeric value of ITOT for the BY group in column 2. The variable EVENT is a character variable, so the numeric value of ITOT is converted to a character value and saved in the character variable ITOTCHAR. Because the IMPACT column is a character column, you can write the string “Tot Event” to it.

```
ODS HTML FILE='TotBreak.html' STYLE=sasweb;
title 'Total At Break';
data _null_;
  set traffic;
  by impact;
  file print ods=(variables=(impact event location highway ));
  length itotchar $ 8;
  if first.impact then do;
     itot = 1;
     put _ods_;
  end;
  else do;
    itot + 1;
    put @2 event @3 location @4 highway;
  end;
  if last.impact then do;
     itotchar=put(itot,8.);
     put @1 'Tot Event' @2 itotchar;
  end;
run;
ODS HTML CLOSE;
```

Figure 9.12 shows the first few observations and the last few observations in the first BY group.

Figure 9.12

Total At Break

Effect on Traffic	Event	Location	Highway
Major Jam	Injury	San Francisco	280
	Injury	San Francisco	80
	Disabled	South Bay	101
	Disabled	San Francisco	101
	Disabled	San Francisco	80
	Minor	San Francisco	280
	Minor	San Francisco	280
	Minor	South Bay	80

. . .

	Injury	San Francisco	280
	Minor	San Francisco	101
	Disabled	San Francisco	280
Tot Event	91		
Slowdown	Disabled	South Bay	80
	Fatal	San Francisco	280
	Fatal	San Francisco	280
	Fatal	San Francisco	80

Applying Inline Formatting

This example enhances the previous example by applying bold font to the relevant columns to call attention to the first event and the summary data line. This is done by concatenating inline formatting to each variable value. When the DATA step writes out this new value, it applies the embedded formatting instruction to the original value.

The lengths of the IMPACT and EVENT variables need to be increased to contain the several characters in the style attribute specification as well as each variable's value. A LENGTH statement before the SET statement defines new lengths for these two variables and the ITOTCHAR variable. Although the DATA step does not apply formatting to the values of EVENT, it does write the value of ITOTCHAR in the same column as the values of EVENT, so both variables need to be increased in length.

The DATA step applies the following formatting to the report:

- It centers the values of HIGHWAY.
- It writes the value of IMPACT in bold for the first observation in a BY group.
- It writes the text string "Tot Event" and the number of events (ITOTCHAR) in bold for the last observation in a BY group.
- It right-justifies the number of events (ITOTCHAR).

```
ODS HTML file='UseEsc.html' STYLE=sasweb;
    ODS PDF file='UseEsc.pdf';
    ODS RTF file='UseEsc.rtf';
    ODS ESCAPECHAR='^';
    title 'DATA Step with Inline Formatting';
    data _null_;
      length impact event highway itotchar $ 50;
      set traffic;
      by impact;
      file print ods=(variables=(impact event location highway));
      highway = cats('^{style[just=c]',highway,'}');
      if first.impact then do;
        itot = 1;
        impact = cats('^{style[font_weight=bold]',impact,'}');
        put _ods_;
      end;
      else do;
        itot + 1;
        put @2 event @3 location @4 highway;
      end;
      if last.impact then do;
        itotchar=
        cats('^{style[just=r font_weight=bold]',left(put(itot,8.)),'}');
        put @1 '^{style[font_weight=bold]Tot Event}' @2 itotchar;
    end;
    run;
    ODS _ALL_ CLOSE;
```

Although the justification style attribute does not apply in the HTML destination, it does in the PDF and RTF destinations. Figure 9.13 shows HTML output, and Figure 9.14 shows PDF output.

Figure 9.13: HTML Output

DATA Step with Inline Formatting

Effect on Traffic	Event	Location	Highway
Major Jam	Injury	San Francisco	280
	Injury	San Francisco	80
	Disabled	South Bay	101
	Disabled	San Francisco	101

. . .

	Minor	San Francisco	101
	Disabled	San Francisco	280
Tot Event	**91**		
Slowdown	Disabled	South Bay	80
	Fatal	San Francisco	280

Figure 9.14: PDF Output

DATA Step with Inline Formatting

Effect on Traffic	Event	Location	Highway
Major Jam	Injury	San Francisco	280
	Injury	San Francisco	80
	Disabled	South Bay	101
	Disabled	San Francisco	101

. . .

	Minor	San Francisco	101
	Disabled	San Francisco	280
Tot Event	**91**		
Slowdown	Disabled	South Bay	80
	Fatal	San Francisco	280

Using FILE PRINT Options and ODS

No discussion of the FILE PRINT ODS statement would be complete without a few caveats. Some of the options that you might have used in the past work only in the LISTING destination.

You should *not* use the following six options with FILE PRINT ODS:

DELIMITER=
DSD
FILE=
FILEVAR=
HEADER=
PAD

The following options in the FILE PRINT statement work *only* in the LISTING destination:

TITLES and NOTITLES
FOOTNOTES and NOFOOTNOTES
LINESIZE=
PAGESIZE=

In addition, some other techniques such as PUT _PAGE_ work only in the LISTING destination. Although column pointer controls such as + and / do work with ODS in the DATA step, they work only within the constraints already discussed—where you are writing to a specific table column in a table instead of to a single print position in a grid.

Exciting changes are on the way for ODS and the DATA step. For example, object-dot syntax features are planned, which would enable you to define report regions, instantiate ODS objects, and place data set information in the regions. Refer to the SAS Web site for more information.

Chapter 10
Creating Custom Reports from the DATA Step

In the previous chapter, you saw how to use the FILE PRINT ODS and PUT _ODS_ statements to write reports from DATA steps to ODS destinations. The DATA steps in that chapter used a default table definition, or template, when they sent reports to ODS. You can also create customized ODS output with the DATA step by specifying the table template that ODS should use to layout the report.

Table templates are an integral part of ODS. Output objects, which are the basic building blocks of SAS procedure output, are created when a procedure takes your results and binds the results to the table template for the procedure. Every ODS TRACE entry in the SAS log contains a reference to the table template that was used to create a particular output object.

This chapter shows you how to harness the power of table templates with DATA step programs so that you can create customized tables that you can send to ODS destinations.

This chapter uses the data sets HR and SALESDATA to illustrate the concepts. These data sets are available on the author pages for this book at http://support.sas.com/authors.

If you are new to the TEMPLATE procedure, it might be helpful to review the basic examples in Chapter 13, “Style Template Definitions,” before continuing with this chapter.

Understanding the Table Definition Used by the DATA Step

When you use the FILE PRINT ODS and PUT _ODS_ statements, you are actually using the default Base.Datastep.Table template, or table definition. This template defines just two generic columns: _NUMVAR_ and _CHARVAR_. These generic columns receive the variable values from the DATA step. Here is the complete definition of the Base.Datastep.Table template.

```
proc template;
  define table Base.Datastep.Table;
    notes "Default DATA Step Table Definition";
    column _numvar_ _charvar_;
    define _numvar_;
      define header n_hdr;
        text _label_;
        just = c;
      end;
      header = n_hdr;
      generic;
    end;
    define _charvar_;
      define header c_hdr;
        text _label_;
        just = c;
      end;
      header = c_hdr;
      generic;
    end;
    justify;
    order_data;
  end;
run;
```

The two variables, _NUMVAR_ and _CHARVAR_, have essentially the same header definition, but they have different names: N_HDR and C_HDR. Each header definition starts with the DEFINE HEADER statement. The column header text that ODS uses is the label of the variable that is being listed, which is specified with the symbolic name _LABEL_. If the variable does not have a label, ODS uses the variable name as the header text. Both definitions center the header text with the JUST=C attribute.

What makes this table definition so flexible is the GENERIC attribute for each column definition. The GENERIC attribute “turns on” the ability to reuse the column definition in which it appears. So the template uses the _NUMVAR_ generic definition for all the numeric variables requested in the report and it uses the _CHARVAR_ definition for all the character variables requested in the report. When you are using a custom table template, the GENERIC column attribute is set to OFF, unless you explicitly turn it on. -

One thing that you can learn from the default table template for the DATA step is that the

ODS developers have made the syntax very flexible. For example, the GENERIC attribute could be coded alternatively as GENERIC=ON or GENERIC=YES, but the simple GENERIC also suffices. The ability to specify just the attribute name, or *attribute*=ON, is true of any of the ON/OFF attributes. Consult the documentation to determine the default attribute values for the attributes of interest to you.

The DATA step default table template uses two other attributes: the ORDER_DATA attribute and the JUSTIFY attribute. Notice how these two attributes are not placed within either of the column definitions. This placement means that they are *table* attributes and not *column* attributes. They apply to the whole table.

The ORDER_DATA attribute is especially useful when you are using generic column definitions. If the ORDER_DATA table attribute is set to ON, then the order of the variables in the Program Data Vector (PDV) or the order of the variables in the VARIABLE= suboption determine the order that the variables have in the destination file. When the ORDER_DATA table attribute is set to OFF, which is the default, then the order of the variables in the template COLUMN statement would determine the order of the variables or columns in the destination file. If your template definition does not have a COLUMN statement, then the order in which you define columns in the template controls the order of the variables in the output.

The JUSTIFY table attribute affects whether the formatted width is taken into account when the variable is written. Essentially, JUSTIFY=OFF says to respect the total formatted width and it turns off the ability to justify. The setting JUSTIFY=ON says to ignore the formatted width, and thus allow justification to happen according to the JUST= attribute.

You can see the effect of JUSTIFY=ON or OFF in one of the custom table templates in the last example in this chapter. All you have to remember is that several pieces of SAS and ODS work together to determine the justification of a variable in the output table: the variable's type and format width, the value for the JUST= column attribute, and the value for the JUSTIFY= table attribute.

Just a few parting notes about custom table template definitions. The BASE table template previously discussed is stored in the SASHELP.TMPLMST template store along with the procedure table templates, the ODS style templates, the ODS MARKUP tagset templates, and the ODS GRAPHICS graph templates. When you modify any template, if you do not specify the new location in an ODS PATH statement, the template is written to the SASUSER.TEMPLAT item store. The examples in this chapter use the following ODS PATH statement to write the custom table templates to the WORK library:

```
ODS PATH WORK.NEWTEMP(UPDATE) SASHELP.TMPLMST(READ);
```

It is a good idea to always include the SASHELP.TMPLMST item store in your ODS PATH statement so subsequent steps can find the SAS production templates.

To create custom templates for others to use, store them in a permanent SAS library, and then everyone who wants to access those templates can point to the item store with an ODS PATH statement.

Creating a Custom Table Template with a DATA Step

In this first example, we design the custom table template HR_TABLE based on two variables from the HR data set and on two variables calculated in the DATA step that creates a report. The template is stored in the TABLES folder in the WORK.NEWTEMP item store. Here's the template:

```
ODS PATH WORK.NEWTEMP(UPDATE) SASHELP.TMPLMST(READ);
proc template;
  define table tables.hr_table;
  column Department AnnualSalary BonusAmt NewSal;
    define Department;
      header='Dept';
    end;
    define AnnualSalary;
      header='Annual';
      format=dollar14.2;
    end;
    define BonusAmt;
      header = '#Bonus#@5%';
      format=7.2;
    end;
    define NewSal;
      header = 'Adjusted';
      format=dollar14.2;
    end;
  end;
run;
```

This simple template specifies attributes for the headers for each column in the report table. The HEADER= option specifies the header text for the column. The FORMAT= option specifies the format for the values in the column. The pound signs (#) surrounding the text "Bonus" in the definition for BONUSAMT causes the text "Bonus" to be on a separate line from the text "@5%".

The next step is to apply the custom template to a DATA step report. The following DATA step calculates two new variables, BONUSAMT and NEWSAL. It creates a report using the layout defined by the HR_TABLE template. The TEMPLATE= suboption in the ODS= option of the FILE PRINT ODS statement requests the TABLES.HR_TABLE template.

```
ODS HTML FILE='CustomHRLayout.html' STYLE=sasweb;
title 'Custom Layout with the DATA Step';
data _null_;
  set hr;
  BonusAmt = AnnualSalary * .05;
  NewSal = AnnualSalary + BonusAmt;
  file print ods=(template='tables.hr_table');
  put _ods_;
run;
ODS HTML CLOSE;
```

Figure 10.1 shows the results sent to the LISTING destination and Figure 10.2 shows the

HTML results.

Figure 10.1

Custom Layout with the DATA Step

Dept	Annual	Bonus @5%	Adjusted
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$64,500.00	3225.00	$67,725.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Administration	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00

Figure 10.2

Custom Layout with the DATA Step

Dept	Annual	Bonus @5%	Adjusted
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$64,500.00	3225.00	$67,725.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Administration	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00

Calculating Columns in a Custom Table Template

In the previous example, the DATA step calculated the new variables. This example shows you how to calculate the new columns in the template definition itself. It is as simple as adding a COMPUTE statement within the DEFINE block for each column. Only the changed DEFINE blocks are shown in the following code. This definition assigns the name HR_CALC to the template.

```
ODS PATH WORK.NEWTEMP(UPDATE) SASHELP.TMPLMST(READ);
proc template;
  define table tables.hr_calc;
  column Department AnnualSalary BonusAmt NewSal;
. . . more code . . .
    define BonusAmt;
      compute as AnnualSalary * .05;
      header = '#Bonus#@5%';
      format=7.2;
    end;
    define NewSal;
      compute as AnnualSalary + BonusAmt;
      header = 'Adjusted';
      format=dollar14.2;
    end;
. . .more code . . .
```

You can use any calculation, function, or equation in the COMPUTE-AS statement that is valid in the SAS language WHERE statement or WHERE clause. Now the DATA step only has to pass the values of the DEPARTMENT and ANNUALSALARY variables to template HR_CALC, and the template calculates the values of columns BONUSAMT and NEWSAL.

```
ODS HTML FILE='CalcHRLayout.html' STYLE=sasweb;
title 'Columns Calculated in Custom Template';
data _null_;
  set hr;
  file print ods=(template='tables.hr_calc');
  put _ods_;
run;
ODS HTML CLOSE;
```

Figure 10.3 shows the results in the Output window and Figure 10.4 shows the HTML results. The results are the same as those shown for the previous example.

Figure 10.3

Columns Calculated in Custom Template

Dept	Annual	Bonus @5%	Adjusted
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$64,500.00	3225.00	$67,725.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Administration	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00

Figure 10.4

Columns Calculated in Custom Template

Dept	Annual	Bonus @5%	Adjusted
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$64,500.00	3225.00	$67,725.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Administration	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00

Creating Spanning Column Headers in a Custom Table Template

You've already seen how custom headers can be defined for each column in a table template definition. In a similar fashion, you can define a header for the whole table or for multiple adjacent columns within a table by placing a DEFINE HEADER section inside the overall table template definition and outside of the column definitions over which the header should span.

The template HR_SPAN below defines two spanning headers: SPANHEAD and CALCHEAD. The SPANHEAD text spans the columns in the COLUMN statement starting from ANNUALSALARY and ending with NEWSAL. The CALCHEAD text spans the two columns BONUSAMT and NEWSAL. If you do not specify START= or END= attributes in a header definition block, the header then spans the entire table.

The order of the header items in the HEADER statement determine the order that the spanning headers are placed above the table (or, if in a FOOTER statement, below the table). This template places the SPANHEAD text above the CALCHEAD text.

```
ODS PATH WORK.NEWTEMP(UPDATE) SASHELP.TMPLMST(READ);
proc template;
  define table tables.hr_span;
  column Department AnnualSalary BonusAmt NewSal;
  header spanhead calchead;
    define header spanhead;
      text 'Based on Salary';
      just=center;
      start=AnnualSalary;
      end = NewSal;
    end;
    define header calchead;
      text 'Calculated';
      just = center;
      start = BonusAmt;
      end = NewSal;
    end;
    . . . rest of template definition as in previous example . . .
  end;
run;
```

The DEFINE blocks for the variables are the same as in the previous example, where the template definition included calculations for BONUSAMT and NEWSAL. The DATA step that calls the HR_SPAN template follows:

```
data _null_;
  set hr;
  file print ods=(template='tables.hr_span');
  put _ods_;
run;
```

Figure 10.5 shows the HTML output.

Figure 10.5

Spanning Columns in Custom Template

Dept	Based on Salary		
		Calculated	
	Annual	Bonus @5%	Adjusted
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$64,500.00	3225.00	$67,725.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Administration	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00

Creating a Custom HTML Table Definition

If you know that you are always going to use HTML for your ODS results file, then you can use a custom table template that is tailored for HTML use. Table template syntax allows you to apply style attributes such as PREHTML=, POSTHTML=, HTMLCLASS=, and TAGATTR= to your HTML report. These attributes are designed for use exclusively in HTML output. You can specify style attributes for either the whole table or for a column header. When you use HTML tags inside ODS, you generally have to turn the PROTECTSPECIALCHARS attribute to OFF in the PROC TEMPLATE STYLE= statement. This is done so that HTML interprets the tag delimiters (< >).

The template HR_HTML defined below specifies header-spanning text and calculates column values as in the previous example. It also modifies the bottom borders of the spanning text and inserts an image above the DEPARTMENT column. The STYLE= statement turns off the interior table lines with RULES=NONE and CELLSPACING=0 so that you can really see the effects of the border style set with the HTML STYLE attribute passed to the HTML file with the ODS TAGATTR attribute.

```
ODS PATH WORK.NEWTEMP(UPDATE) SASHELP.TMPLMST(READ);
proc template;
  define table tables.hr_html;
  column Department AnnualSalary BonusAmt NewSal;
  style={rules=none frame=box cellspacing=0
        protectspecialchars=off};
  header spanhead calchead;
    define header spanhead;
      text 'Based on Salary';
      just=center;
      start=AnnualSalary;
      end = NewSal;
      style={tagattr='style="border-bottom:thick double black"'};
    end;
    define header calchead;
      text 'Calculated';
      just=center;
      start=BonusAmt;
      end = NewSal;
      style={tagattr='style="border-bottom:thick double black"'};
    end;
    define Department;
      define header dhead;
        text 'Dept';
        style={prehtml='<img src="LOGO.GIF">'};
      end;
      header=dhead;
    end;
    . . . rest of template definition as in previous example . . .
run;
```

With the exception of the TEMPLATE= suboption value specified as TABLES.HR_HTML, the same DATA step as in the previous example is submitted. Figure 10.6 shows the HTML output.

Figure 10.6

HTML Custom Report

LH Dept	Based on Salary		
		Calculated	
	Annual	Bonus @5%	Adjusted
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$64,500.00	3225.00	$67,725.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Administration	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00

Performing Traffic Lighting in a Custom Table Template

Traffic lighting is a technique that uses fonts and colors to call attention to particular results within a larger table. Two ways to specify traffic lighting in output created from a custom table template are available to you. Just as with PROC PRINT, PROC REPORT, and PROC TABULATE, you can perform traffic lighting through user-defined formats that specify style attribute values such as BACKGROUND=, FOREGROUND=, or FONT_FACE=. The second way to perform traffic lighting is with the PROC TEMPLATE statement CELLSTYLE-AS. The CELLSTYLE-AS statement is unique to table templates.

The template HR_TRAFFIC below applies both methods. The column for DEPARTMENT applies format $DFONT. The column BONUSAMT uses a CELLSTYLE-AS statement to identify ranges of bonus amounts. The remainder of the template HR_TRAFFIC is the same as the template HR_CALC, which was defined a few examples back.

```
ODS PATH WORK.NEWTEMP(UPDATE) SASHELP.TMPLMST(READ);
proc template;
  define table tables.hr_traffic;
  column Department AnnualSalary BonusAmt NewSal;
    define Department;
      style={font_face=$DFont.};
    end;
    define BonusAmt;
. . . more code . . .
      cellstyle _val_ between 1600 and 2500 as
                {font_weight=bold},
               _val_ between 2500 and 4000 as
                {background=black foreground=white},
               _val_ gt 4000 as {font_style=italic},
                1 as Data;
    end;
run;
```

The $DFONT format does not need to be defined when the PROC TEMPLATE step compiles the template definition. It does, however, need to be available when the DATA step references the template in the FILE PRINT ODS statement in the DATA step. The FONT_FACE= attribute can take the name of a valid font value. The PROC FORMAT code is shown below:

```
proc format;
  value $DFont 'Administration','R&D'     ='Courier New'
               'Marketing','Manufacturing'='SAS Monospace';
run;
```

The CELLSTYLE-AS statement evaluates an expression in the same manner as the COMPUTE statement. The expression can be any expression that is valid in a SAS language WHERE statement. The logic of the CELLSTYLE-AS statement operates like nested IF/THEN/ELSE statements in that as soon as the first condition is met, ODS applies the style changes and it does no other evaluation for that cell. _VAL_ is a special symbolic name for

the current value in the cell.

The same DATA step as in the previous example is submitted with the single difference being that the TEMPLATE= suboption value is specified as TABLES.HR_TRAFFIC. Figure 10.7 shows the HTML output. The output sent to the LISTING destination in Figure 10.8 shows that it does not support style changes.

Traffic Lighting in Custom Template

Dept	Annual	Bonus @5%	Adjusted
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$64,500.00	3225.00	$67,725.00
Manufacturing	$114,000.00	*5700.00*	$119,700.00
Administration	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$114,000.00	*5700.00*	$119,700.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$46,000.00	2300.00	$48,300.00
Manufacturing	$68,500.00	3425.00	$71,925.00
R&D	$49,000.00	2450.00	$51,450.00
Administration	$126,500.00	*6325.00*	$132,825.00

Figure 10.7

Figure 10.8

Traffic Lighting in Custom Template

Dept	Annual	Bonus @5%	Adjusted
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$64,500.00	3225.00	$67,725.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Administration	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$114,000.00	5700.00	$119,700.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00
Manufacturing	$35,000.00	1750.00	$36,750.00

Using Generic Columns in a Table Template

Because ANNUALSALARY, BONUSAMT, and NEWAMT are all numeric variables in the data set HR, one approach for a custom table template might be to define a template that has only two items in the COLUMN statement: one for character variable DEPARTMENT and a second, generic column for the three numeric variables. The ability to define generic or reusable column items in your table template is a powerful feature of table templates. As we discussed previously, the default template used with the DATA step uses only two generic columns: one for numeric variables and one for character variables.

```
proc template;
  define table tables.hr_generic;
  column Department SalGen;
  dynamic colhdr;
    define Department;
      header='Dept';
      style=RowHeader;
    end;
    define SalGen;
      header=colhdr;
      generic=on;
    end;
  end;
run;
```

In the DATA step that follows, the COLUMNS= suboption lists the four variables that should appear in the report. The program includes additional specifications for the three numeric variables.

The specification for each numeric variable starts with the name of the generic definition to apply to the variable, followed by an equal sign and the name of the numeric variable. Suboptions to apply to the variable are enclosed in parentheses. The DYNAMIC= suboption defines header text for the variable by assigning text to the dynamic variable COLHDR. The GENERIC=ON suboption is also applied to each numeric variable. The FORMAT statement at the bottom of the DATA step formats the values passed to the template. Because variables BONUSAMT and NEWAMT have different formats, it is not possible to hardcode the formats in the generic column definition.

```
title "Generic Variables in Custom Template";
data _null_;
  set hr;
  BonusAmt = AnnualSalary * .05;
  NewAmt = AnnualSalary + BonusAmt;
  file print ods=
      (template='tables.hr_generic'
       columns=(Department
              SalGen=AnnualSalary
          (dynamic=(colhdr='Annual Salary') generic=on)
              SalGen=BonusAmt
          (dynamic=(colhdr='Bonus') generic=on)
              SalGen=NewAmt
          (dynamic=(colhdr='Adj Salary') generic=on)));
```

```
  put _ods_;
  format BonusAmt comma8.2 NewAmt Dollar14.2;
run;
```

Figure 10.9 shows the HTML output.

Figure 10.9

Generic Variables in Custom Template

Dept	Annual Salary	Bonus	Adj Salary
Manufacturing	$35,000	1,750.00	$36,750.00
Manufacturing	$64,500	3,225.00	$67,725.00
Manufacturing	$114,000	5,700.00	$119,700.00
Administration	$35,000	1,750.00	$36,750.00
Manufacturing	$35,000	1,750.00	$36,750.00
Manufacturing	$114,000	5,700.00	$119,700.00
Manufacturing	$35,000	1,750.00	$36,750.00
Manufacturing	$35,000	1,750.00	$36,750.00
Manufacturing	$35,000	1,750.00	$36,750.00
Manufacturing	$35,000	1,750.00	$36,750.00
Manufacturing	$35,000	1,750.00	$36,750.00
Manufacturing	$46,000	2,300.00	$48,300.00
Manufacturing	$68,500	3,425.00	$71,925.00
R&D	$49,000	2,450.00	$51,450.00

Using Macro Variables in a Custom Table Template

It is possible to use either numeric or character macro variables in a template definition (in addition to dynamic variables). Macro programmers might find it odd to refer to numeric or character macro variables, but you have to consider *how* the macro variable will be used in the template.

Depending on the type of the macro variable, you declare the macro variable with either the MVAR or NMVAR statement in PROC TEMPLATE. For example, use the MVAR statement to define a character macro variable that will hold header text, and use the NMVAR statement to define a numeric macro variable that will hold the results of a numeric calculation.

The template HR_MACRO defines four character macro variables and one numeric macro variable. The template inserts the values of the character macro variables into header text. The numeric macro variable value is used in the calculation of BONUSAMT.

The macro variables defined in a template are actual SAS macro variables that you or SAS defines outside the template definition. Before executing the DATA step that invokes the template definition, you need to define the user-defined macro variables and assign them values. This is shown in the program following the PROC TEMPLATE step. It is not necessary to create the user-defined macro variables before compiling the table definition.

```
ODS PATH WORK.NEWTEMP(UPDATE) SASHELP.TMPLMST(READ);
proc template;
  define table tables.hr_macro;
  column Department AnnualSalary BonusAmt NewSal;
  mvar thisyear btxt sysdate9 sysscpl;
  nmvar bpct;
  header spanhead;
  footer rundate;
    define header spanhead;
      text 'Based on ' thisyear
           ' Salary Numbers at ' btxt '%';
      just=center;
    end;
    define footer rundate;
      text 'Run on: ' sysdate9 ' Platform: ' sysscpl;
    end;
    define BonusAmt;
      compute as AnnualSalary * bpct;
      header = "Bonus";
      format=6.2;
    end;
    define NewSal;
      compute as AnnualSalary + BonusAmt;
      header = 'Adjusted';
      format=dollar14.2;
    end;
  end;
run;
```

The following program starts with three %LET statements that define the three user-defined macro variables and assign them values. The other two macro variables, SYSDATE and SYSSCPL, are system-defined automatic macro variables. Therefore, you do not need to assign values to SYSDATE and SYSSCPL, because SAS defines them and assigns them values when your SAS session starts. They do, however, have to appear in the MVAR statement so that the template knows to look for them in the Global Symbol Table.

```
%let thisyear = 2008;
%let bpct = 0.075;
%let btxt = 7.5;
ODS HTML FILE='HRMacroVars.html' style=sasweb;
title 'Using Macro Variables in Custom Template';
data _null_;
  set hr;
  file print ods=(template='tables.hr_macro');
  put _ods_;
run;
ODS HTML CLOSE;
```

Figure 10.10 shows the HTML output.

Figure 10.10

Using Macro Variables in Custom Template

Based on 2008 Salary Numbers at 7.5%			
Department	Annual Salary	Bonus	Adjusted
Manufacturing	$35,000	2625.0	$37,625.00
Manufacturing	$64,500	4837.5	$69,337.50
Manufacturing	$114,000	8550.0	$122,550.00
Administration	$35,000	2625.0	$37,625.00
. . .			
Manufacturing	$35,000	2625.0	$37,625.00
R&D	$35,000	2625.0	$37,625.00
Manufacturing	$73,000	5475.0	$78,475.00
Marketing	$35,000	2625.0	$37,625.00
Run on: 27NOV2008 Platform: XP_PRO			

Using Dynamic Variables as Attribute Values in a Custom Table Template

This example demonstrates the effects of the JUSTIFY= attribute by reporting on a range of numeric data of varying sizes and precision from the data set SALESDATA. This small data set of sales, commission, and profit data is shown in Figure 10.11. The template TABLE_JUST defined in the following code uses the variable names from SALESDATA for the template column names, and it bypasses using GENERIC column definitions for now.

The JUSTIFY= attribute affects whether the formatted width is taken into account when the variable is written. Essentially, JUSTIFY=OFF says to respect the total formatted width and it turns off the ability to justify. The setting JUSTIFY=ON says to ignore the formatted width, and thus allow justification to happen according to the JUST= attribute.

```
ODS PATH WORK.NEWTEMP(UPDATE) SASHELP.TMPLMST(READ);
proc template;
  define table tables.table_just;
  column Type Sales AmtCom Profit PctCom;
    define Type;
      header='Type';
    end;
    define Sales;
      header='#Sales#Amount#c/off';
      just=c;
      justify=off
      format=dollar14.2;
    end;
    define AmtCom;
      header = '#Amount#Commission#c/on';
      just=c;
      justify=on;
      format=dollar14.2;
    end;
    define Profit;
      header = '#Profit#l/off';
      just=l;
      justify=off;
      format=dollar14.2;
    end;
    define PctCom;
      header = '#Pct#Commission#l/on';
      just=l;
      justify=on;
      format=6.2;
    end;
  underline=on;
  end;
run;
ODS LISTING;
ODS PDF file='Table_Just.pdf';
ODS RTF file='Table_Just.rtf';
ODS HTML file='Table_Just.html' style=sasweb;
title 'JUST= and JUSTIFY= Attributes in Custom Table Template';   data _null_;
```

```
  set salesdata;
  if Sales lt 1 then AmtCom = (Sales * -1) * PctCom;
  else if Sales ge 0 then AmtCom = Sales * PctCom;
  Profit = Sales - AmtCom;
  file print ods=(template='tables.table_just');
  put _ods_;
run;
ODS _ALL_ CLOSE;
```

Because we put the values of the JUST= attribute and the JUSTIFY= attribute in the column headers, it is easy to see how the attribute values were used to display the numbers. Column SALES has JUST=C and JUSTIFY=OFF. The output shows that the formatted width was respected and that the values of SALES were not centered. On the other hand, column AMTCOM column has JUST=C and JUSTIFY=ON. Its formatted width was ignored, and center justification was enabled.

Another table attribute that you see in the table template is the UNDERLINE= attribute. This attribute caused the underlining of the headers in the output sent to the LISTING destination, as seen in Figure 10.12. Figure 10.13, however, shows that ODS does not underline the headers in the output for the PDF destination (and does not for RTF and HTML output either).

To make the template more flexible, you can change the way you supply values to the JUST= and JUSTIFY= attributes by using dynamic variables and macro variables. Most attributes will take a dynamic variable value or macro variable value as an attribute value. Here, we define the dynamic variables JVAL and Y_OR_N. If you add the following DYNAMIC statement to your template, you can reference the two dynamic variables in the column definitions.

```
dynamic jval y_or_n;
```

The following JUST= and JUSTIFY= attributes reference JVAL and Y_OR_N.

```
just=jval;
justify=y_or_n;
```

The final piece is to assign a value to the dynamic variables in the DATA step as shown in the following code. For each column, the value assigned to dynamic variable JVAL passes to the template and sets the JUST= attribute value. Similarly, the value assigned to dynamic variable Y_OR_N sets the JUSTIFY= attribute for each column.

```
data _null_;
  set salesdata;
  AmtCom = Sales * PctCom;
  Profit = Sales - AmtCom;
  file print
      ods=(template='tables.table_jval'
          columns=(
          Type(dynamic=(jval='r' y_or_n=0))
          Sales(dynamic=(jval='c' y_or_n=1))
```

```
            AmtCom(dynamic=(jval='c' y_or_n=0))
            Profit(dynamic=(jval='l' y_or_n=1))
            PctCom(dynamic=(jval='l' y_or_n=0))
           ));
  put _ods_;
run;
```

Submitting this program generates the same results as the previous program. The only difference is that the values assigned to JVAL and Y_OR_N control the justification of the columns.

In the previous chapter, you used the VARIABLES= option with the default table template to control the order of the variables in the result output. When you use a custom table template and the TEMPLATE= option, you need to use the COLUMNS= option to specify the variable names, and optionally to provide any suboptions, such as DYNAMIC= or LABEL= or FORMAT=.

We revisit the use of DYNAMIC variables along with MVAR or NMVAR variables in Chapter 20, "Understanding ODS Graphics Output," in the discussion of Graph templates.

Figure 10.11

Obs	Type	Sales	Pct Com	by
1	TEL	10	0.02	cz
2	TEL	200	0.07	lh
3	WEB	3	0.02	pt
4	WEB	40	0.03	pt
5	TEL	5	0.02	lh
6	TEL	-15	0.02	return
7	WEB	70	0.03	cz
8	TEL	800	0.10	lh
9	WEB	90	0.05	cz

Figure 10.12

JUST= and JUSTIFY= Attributes in Custom Table Template

Type	Sales Amount c/off	Amount Commission c/on	Profit l/off	Pct Commission l/on
TEL	$10.00	$0.20	$9.80	0.02
TEL	$200.00	$14.00	$186.00	0.07
WEB	$3.00	$0.06	$2.94	0.02
WEB	$40.00	$1.20	$38.80	0.03
TEL	$5.00	$0.10	$4.90	0.02
TEL	$-15.00	$0.30	$-15.30	0.02
WEB	$70.00	$2.10	$67.90	0.03
TEL	$800.00	$80.00	$720.00	0.10
WEB	$90.00	$4.50	$85.50	0.05

Figure 10.13

JUST= and JUSTIFY= Attributes in Custom Table Template

Type	Sales Amount c/off	Amount Commission c/on	Profit l/off	Pct Commission l/on
TEL	$10.00	$0.20	$9.80	0.02
TEL	$200.00	$14.00	$186.00	0.07
WEB	$3.00	$0.06	$2.94	0.02
WEB	$40.00	$1.20	$38.80	0.03
TEL	$5.00	$0.10	$4.90	0.02
TEL	$-15.00	$0.30	$-15.30	0.02
WEB	$70.00	$2.10	$67.90	0.03
TEL	$800.00	$80.00	$720.00	0.10
WEB	$90.00	$4.50	$85.50	0.05

Chapter 11
Modifying and Replaying ODS Output

New in SAS®9 is the ODS DOCUMENT destination and the DOCUMENT procedure, which is the companion procedure intended for use with the DOCUMENT destination. This chapter describes both the DOCUMENT destination and how to use PROC DOCUMENT to create, manage, and replay items sent to this destination.

An ODS document is a special kind of proprietary SAS file that contains the results of your analyses *before* the output has been sent to the final result file. Think of an ODS document as being a pause in the ODS process. You might want to divert procedure output into an ODS document for several reasons.

You might want to separate the time it takes to run the analysis from the time it takes to send the analysis results to the ODS result file. By doing this, you can produce reports in the future from the analysis, and you can also send the results to different destinations. Or you can print only certain components of the analysis without rerunning the lengthy analysis every time you want a new copy of the report.

You might want to reorganize the result hierarchy in another form (the result hierarchy is what you see in the Results window or in any ODS-generated table of contents). For example, you can organize your output by city or other BY-group variable, instead of in procedure order.

You might need to create a permanent record of an entire project's analysis, and you need more than just a paper archive or a disk backup. A requirement is to be able to replay the results in different destinations such as RTF, HTML, or PDF without rerunning the program.

This chapter uses the data set HR to illustrate the concepts. This data set is available on the author pages for this book at http://support.sas.com/authors.

The ODS DOCUMENT Destination

An ODS document store is a special kind of SAS file known as an *item store*. It is a proprietary SAS format and can be viewed and managed only with SAS software. If you work with style and table templates or the SAS registry, you know that those SAS files are also kept in item stores.

Let's start with code that illustrates the basic "sandwich" technique to create an ODS document. The ODS DOCUMENT statement names the document store with the NAME= option. It does not use the FILE= option. The specification for the NAME= option follows the usual SAS file rules, just as if you were creating a new data set. The example saves the document store temporarily in the WORK library. You can save a document store in a permanent location by specifying the library's libref. The WRITE option in the ODS DOCUMENT statement specifies both Read and Write access to the document store.

After the ODS document has been created, PROC DOCUMENT can display information about the document.

```
ODS DOCUMENT NAME=work.origdoc(write);
  proc means data=hr min mean max;
    title 'Salary Analysis by Country';
    by Country;
    class Category;
    var AnnualSalary;
  run;
  proc freq data=hr;
    title 'Department Information by Country';
    by Country;
    table Department;
  run;
ODS DOCUMENT CLOSE;
```

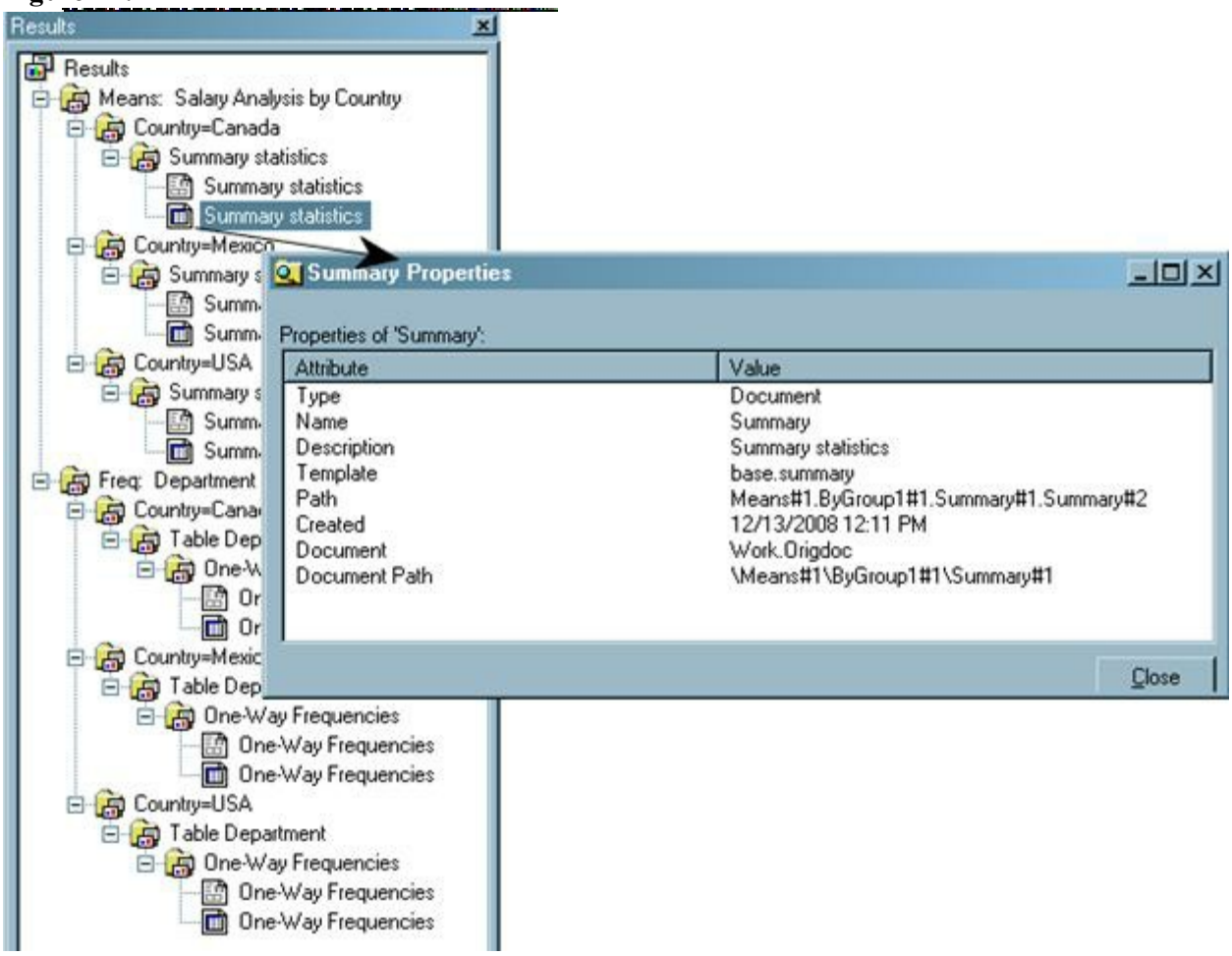
With other ODS destinations, this "sandwich" method sends the results to a final result file. This example instead creates an ODS document store. If you view the PROC CONTENTS or PROC DATASETS output for the WORK library or review the icons in the SAS Results window, you can see that the new ODS document store has been created and that its member type is ITEMSTOR. Figure 11.1 shows part of the PROC DATASETS output for the WORK library. Figure 11.2 shows the Results window and the properties for one output object in the document store. Notice the new icon for an ODS document in the Results windows.

Note that the SAS Explorer window does not show WORK.ORIGDOC. To see the ODS document store, you have to open the ODS Documents window.

Figure 11.1

#	Name	Member Type	File Size	Last Modified
1	FORMATS	CATALOG	17408	13Dec08:12:04:12
2	HR	DATA	33792	13Dec08:12:11:45
3	ORIGDOC	ITEMSTOR	33792	13Dec08:12:11:54

Figure 11.2



Using the Documents Window

Understanding how the ODS Documents window works is easier once you have a document to explore.

To open the window, type **`odsdocument`** in the SAS command line, as shown on the left in Figure 11.3. Alternatively, from the **View** menu, select **Documents**, as shown on the right in Figure 11.3.

Figure 11.4 shows that when the Documents window opens, SAS docks it in the same area as the Results and SAS Explorer windows. You can switch between the three windows by clicking on the appropriate tab at the bottom.

What you see in the ODS Documents window resembles the Results window in that both have a hierarchical structure. However, the Documents window has no output other than the output objects in ODS document form.

After you have created an ODS document store, it is very easy to see the document contents or replay the document to a destination using PROC DOCUMENT. The following examples explore this procedure's usage.

Figure 11.3

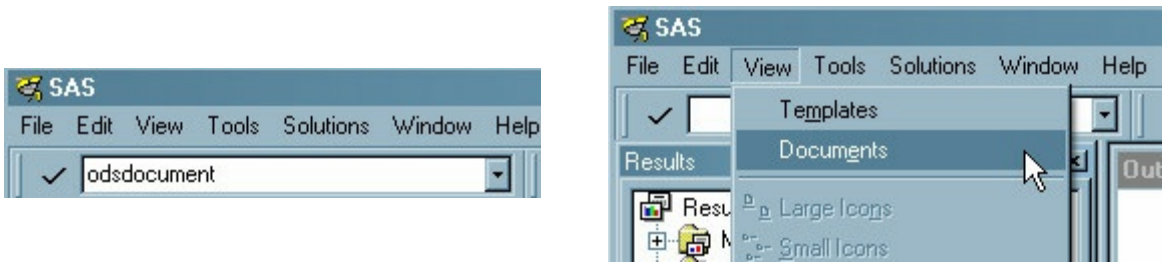
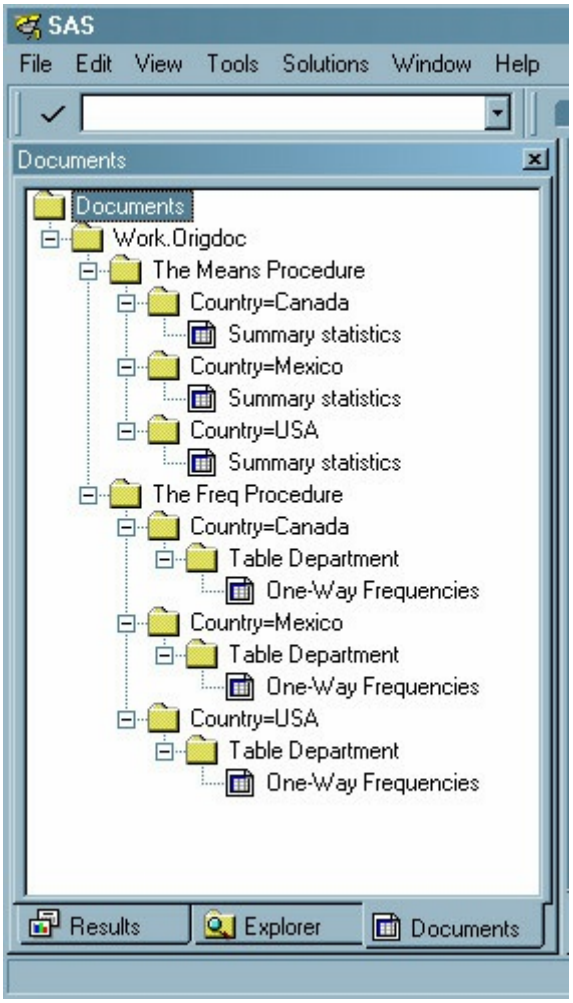


Figure 11.4


SAS
File Edit View Tools Solutions Window Help
Documents
Documents
Work.Origdoc
The Means Procedure
Country=Canada
Summary statistics
Country=Mexico
Summary statistics
Country=USA
Summary statistics
The Freq Procedure
Country=Canada
Table Department
One-Way Frequencies
Country=Mexico
Table Department
One-Way Frequencies
Country=USA
Table Department
One-Way Frequencies
Results
Explorer
Documents

Using PROC DOCUMENT to Manage Document Stores

The first example in this chapter created the document store WORK.ORIGDOC. Now that we have a document store, we can use PROC DOCUMENT to manage it. This procedure can list contents information about the document store as well as rearrange, duplicate, or remove output from it.

The following PROC DOCUMENT step produces a list of the contents of the WORK.ORIGDOC document store. Figure 11.5 shows the output.

```
ODS LISTING;
  proc document name=work.origdoc;
    list / levels=all;
  run;
  quit;
```

If you wanted to see only the list of BY groups, starting in SAS 9.2 you can add the BYGROUPS option to the LIST statement.

```
LIST / LEVELS=ALL BYGROUPS;
```

An important feature of PROC DOCUMENT is that it allows run-group processing. Therefore, the step boundary for a PROC DOCUMENT step is QUIT and not RUN.

In batch mode, as with SAS/GRAPH and some SAS/STAT procedures, it is good coding practice to explicitly code the QUIT statement.

A later example demonstrates how to rearrange the contents of a document store.

Figure 11.5

Output - (Untitled)

Listing of: \Work.Origdoc\
Order by: Insertion
Number of levels: All

Obs	Path	Type
1	\Means#1	Dir
2	\Means#1\ByGroup1#1	Dir
3	\Means#1\ByGroup1#1\Summary#1	Table
4	\Means#1\ByGroup2#1	Dir
5	\Means#1\ByGroup2#1\Summary#1	Table
6	\Means#1\ByGroup3#1	Dir
7	\Means#1\ByGroup3#1\Summary#1	Table
8	\Freq#1	Dir
9	\Freq#1\ByGroup1#1	Dir
10	\Freq#1\ByGroup1#1\Table1#1	Dir
11	\Freq#1\ByGroup1#1\Table1#1\OneWayFreqs#1	Table
12	\Freq#1\ByGroup2#1	Dir
13	\Freq#1\ByGroup2#1\Table1#1	Dir
14	\Freq#1\ByGroup2#1\Table1#1\OneWayFreqs#1	Table
15	\Freq#1\ByGroup3#1	Dir
16	\Freq#1\ByGroup3#1\Table1#1	Dir
17	\Freq#1\ByGroup3#1\Table1#1\OneWayFreqs#1	Table

Replaying a Document Store Using PROC DOCUMENT and Pop-Up Menus

Using the document store we created in the first example, we can replay its content in different destinations using either PROC DOCUMENT or pop-up menus. The following PROC DOCUMENT program replays the document store to the four destinations that are open when PROC DOCUMENT executes. The four destinations are PDF, RTF, HTML, and LISTING.

The partial listing output from this step is shown in Figure 11.6, along with the Results window view of the REPLAY.

```
ODS PDF FILE='ReplayDoc.pdf';
ODS RTF FILE='ReplayDoc.rtf';
ODS HTML FILE='ReplayDoc.html';
ODS LISTING;
  proc document name=work.origdoc;
    replay;
  run;
  quit;
ODS _ALL_ CLOSE;
```

You can also replay the document using pop-up menus. Right-click on the document name in the Documents window and select **Replay** from the bottom of the pop-up menu to immediately replay the ODS document to any open destinations. You have to scroll down to the bottom of the pop-up menu to see the Replay choice. It is not shown in Figure 11.7, however, since the LISTING destination is the only destination that might be open. You can, however, also select **Open As** to select the destination to which you want to replay the ODS document. The **Open As** choice and the Open As dialog box are shown in Figure 11.7. You might want to choose the Open As method or the code method because you can explicitly choose the destination to use for the replayed output.

Whether you immediately replay the ODS document to another destination or save the document in a permanent storage location, the ability to save and replay the document demonstrates reasons 1 and 3 in the introduction to this chapter. Before we move on to the topic of modifying or customizing the ODS document, we want to explain some fundamental concepts of the ODS DOCUMENT destination.

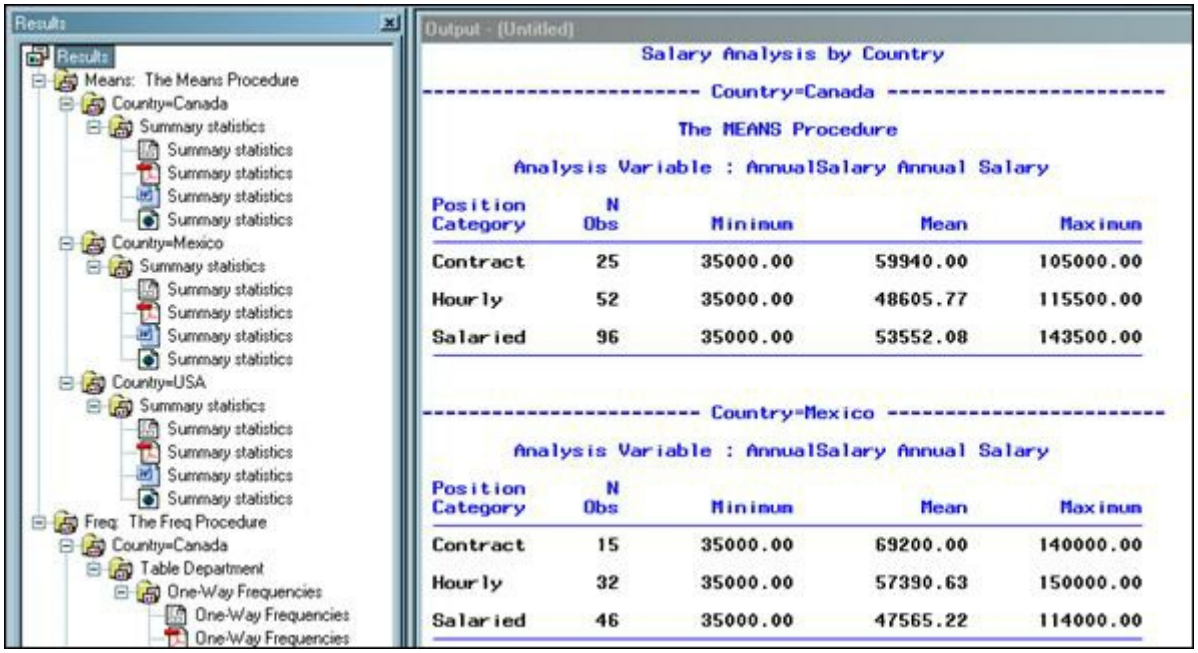
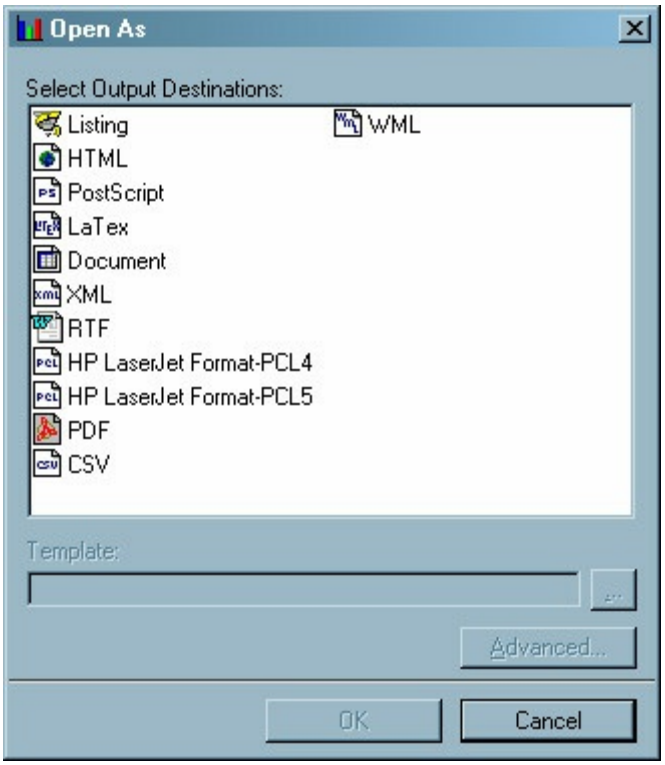


Figure 11.6

Figure 11.7



Understanding the Fundamental Concepts of the ODS DOCUMENT Destination

ODS Document Store Characteristics

An ODS document store is not portable. You cannot move it across platforms. This is true of template item stores, as well as the SAS registry. So although this is a limitation, it is true of all item stores and is not unique to ODS document stores.

The ODS document store is not a SAS data set or catalog. It is actually a hierarchical tree structure of directories (or folders). Figure 11.5 shows several entries with type DIR, which correspond to the folders that you see in the Documents window. The entry type of TABLE is for the output object from the procedure in its raw, unformatted state.

Limited Support of PROC FREQ and PROC PRINT

Three procedures in Base SAS are different from the rest when it comes to ODS because they do not have table templates. The procedures are PRINT, REPORT, and TABULATE.

PROC REPORT and PROC TABULATE are fully supported by ODS DOCUMENT. PROC PRINT output is supported, but with limitations on BY-group processing. PROC PRINT will have corrected ODS DOCUMENT support in SAS 9.3. Although ODS DOCUMENT does support PROC FREQ crosstabulation table output, when replayed the output might look more like a TABULATE table than a classic crosstabulation table.

ODS Document Stores and User-Defined Formats

It is hard to write reports without user-defined formats. Because formats can be stored in either temporary or permanent catalogs, they are problematic for ODS document store storage and management. If you use a temporary format when you create an ODS document store, ODS will save or “persist” the formats used in the creation of the ODS document store.

For example, you might have a temporary format catalog with ten formats defined. But if you use only two of the formats in the creation of the ODS document, only those two formats persist. When you replay the document, the two formats are restored or essentially recreated for use.

Permanent formats, on the other hand, do not persist and they are not saved in the ODS document store. That means any formats that you use which are not stored in the WORK library are considered permanent formats and are not available when you replay the document. If you have a user-defined format you want to use with ODS DOCUMENT, place a copy of it in the WORK library before creating the ODS document store.

ODS Document Stores and SAS/GRAPH Output

Issues with SAS/GRAPH output that is stored in either permanent or temporary GRSEG catalogs are similar to the issues with user-defined formats described in the previous section. To help you save steps and prevent having to recode all jobs with the GOUT= option, you can instead add the CAT= option to your ODS DOCUMENT statement to point to a permanent catalog location. This option essentially instructs ODS to copy any temporary GRSEG catalog entries to the CAT= location. When you use this method of storing and referencing GRSEG catalog entries, you are said to be using an *external graph*.

With some device drivers, you can generate an internal graph as an alternative. This graph type takes advantage of that fact that the client-side device drivers—JAVA, ACTIVEX, JAVAIMG, and ACTXIMG—can store the data points and the template information associated with the output object and create a fully self-describing internal graph. Some limitations exist when using internal graphs. The most important limitation is that you must replay the ODS document with the internal graph using one of these four device drivers, and you must replay the document to a destination that supports the output format that you select.

ODS Document Stores and ODS Templates

Different factors can affect the replay of your output. For example, if you create a document store with a reference to an external template, then ODS looks for that template again when you do the replay. If the template is the same, the output looks as you expect. If, however, the template has been modified, ODS uses the new template definition, and your output might look quite different from what you expect. Or you can take advantage of this feature to generate alternate representations of output objects simply by changing the template between replays. With the statistical graphics (SG) procedure templates, you can even change tables to graphs and vice versa.

It is important to be methodical and either use templates from SASHELP.TMPLMST or remember to save your user-defined templates that were referenced in the creation of your ODS documents. Starting in SAS 9.2, the OBTEMPL statement in PROC DOCUMENT can write the template source code used for an output object to the open destination.

Replaying Document Stores and SAS Options

You also need to pay attention to options that were in place when you created your ODS document store. Having different procedure options, SAS system options, and SAS/GRAPH options in effect when you replay your document compared to when you created your document store can have an effect on the look of your replayed document.

Creating a New Document Store Interactively

The first example in this chapter shows how to create an ODS document store by sending the output from SAS procedures to the DOCUMENT destination. Another way to create a new document store is to use the interactive Documents window. To create a new, empty document store, right-click the **Documents** folder label at the top of the Documents window and select **New Document**. From the New Document dialog box, select the library and name the new document store. Figure 11.8 shows these two steps.

After you have created and named your new document store, you can now define your own folder hierarchy in it. This process is the same as the main folder creation, except when you right-click the name of the new document store, one of your choices is **New Folder**. Again, a dialog box opens in which you name and describe the folder.

The final view of the new document store WORK.NEWHR in Figure 11.9 shows two new folders. Because we chose to provide both a name and a description, the description is what shows in the Documents window. If you don't enter a description, the Documents window displays the name of the store. Note that the name does not have any spaces, although the description can contain spaces.

Figure 11.8

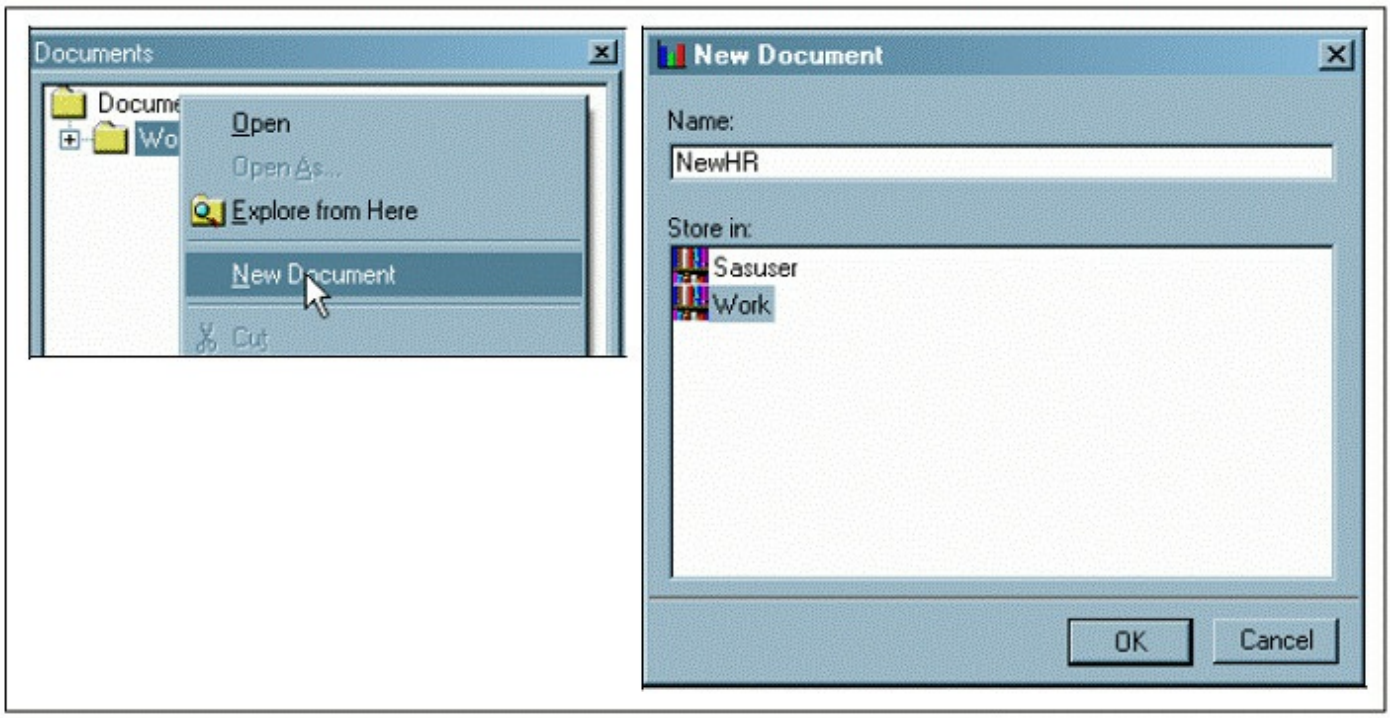
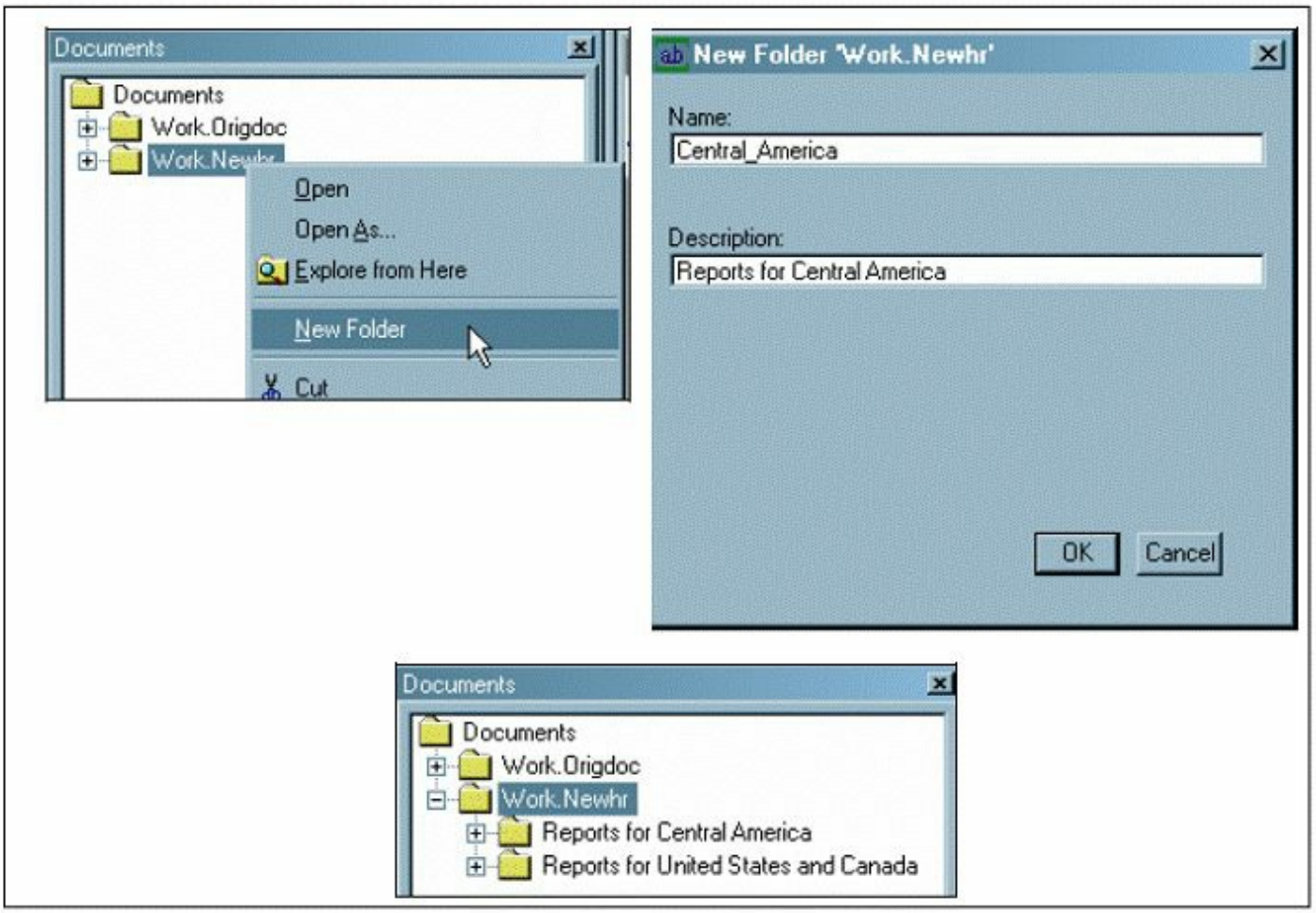


Figure 11.9


Documents
Documents
Work.Origdoc
Work.Newhr
Open
Open As...
Explore from Here
New Folder
Cut
New Folder 'Work.Newhr'
Name:
Central_America
Description:
Reports for Central America
OK
Cancel
Documents
Documents
Work.Origdoc
Work.Newhr
Reports for Central America
Reports for United States and Canada

Creating a New Document Store Interactively (continued)

Figure 11.10 shows the results of adding subfolders underneath the first level of the hierarchy. A folder is added for Canada and a folder for the United States. The general method is the same: right-click on the folder hierarchy, select **New Folder**, and name and describe the document store.

Next you can drag and drop or copy and paste output objects from the original document store to the new document store. It is possible to just rearrange objects in an existing document store, or even link to objects in an original document store. However, we assume that you need to keep a backup copy of the original procedures' output, and you also want copies of the original output objects rearranged in the new document store.

Figure 11.11 shows the results of the copy and paste operation. In addition to arranging our output objects and the folder structure, we also renamed the main folders inside the new document store (by right-clicking the folder name and selecting **Rename**).

Now it's time to replay the new document in the PDF destination. Select the WORK.NEWHR document name, right-click, select **Open As**, and send the replayed document to the PDF destination. Note that the structure of the document is maintained in the Results window on the left in Figure 11.12 and in the PDF bookmark area on the right in Figure 11.12.

Figure 11.10

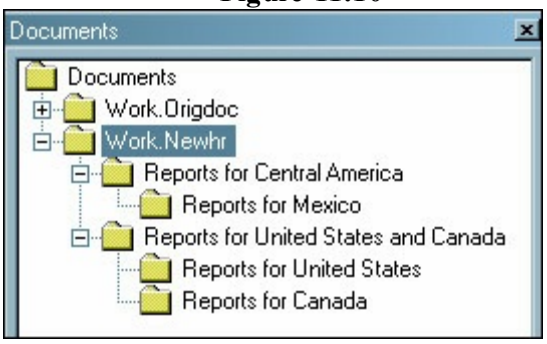


Figure 11.11

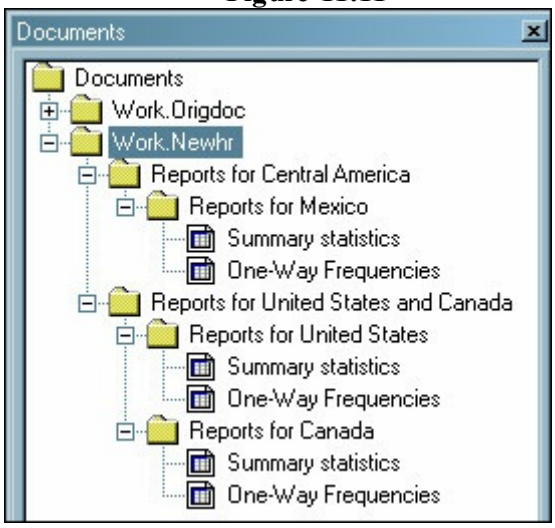
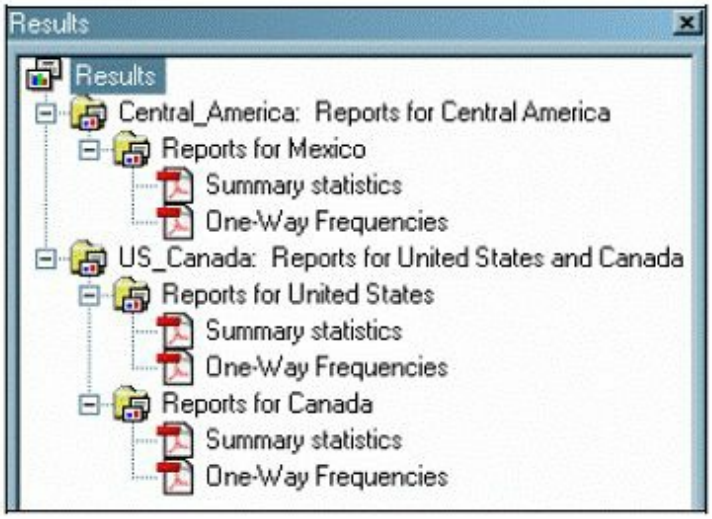


Figure 11.12


Results
Results
Central_America: Reports for Central America
Reports for Mexico
Summary statistics
One-Way Frequencies
US_Canada: Reports for United States and Canada
Reports for United States
Summary statistics
One-Way Frequencies
Reports for Canada
Summary statistics
One-Way Frequencies

Reports for Central America
Reports for Mexico
Summary statistics
One-Way Frequencies
Reports for United States and Canada
Reports for United States
Summary statistics
One-Way Frequencies
Reports for Canada
Summary statistics
One-Way Frequencies

Creating a New Document Store Interactively (continued)

Although the results look good in these views, a closer examination of the output reveals a few things that need to be tweaked. For example, the PDF file created by replaying the original output in WORK.ORIGDOC contains four pages, as shown on the left in Figure 11.13. The three PROC MEANS outputs are on the first page, and each of the PROC FREQ tables is on a separate page.

In contrast, the replayed WORK.NEWHR ODS document has five pages, as shown on the right in Figure 11.13. The PROC FREQ output for Canada and the PROC MEANS output for the United States are on the same page. All the other output objects are on separate pages.

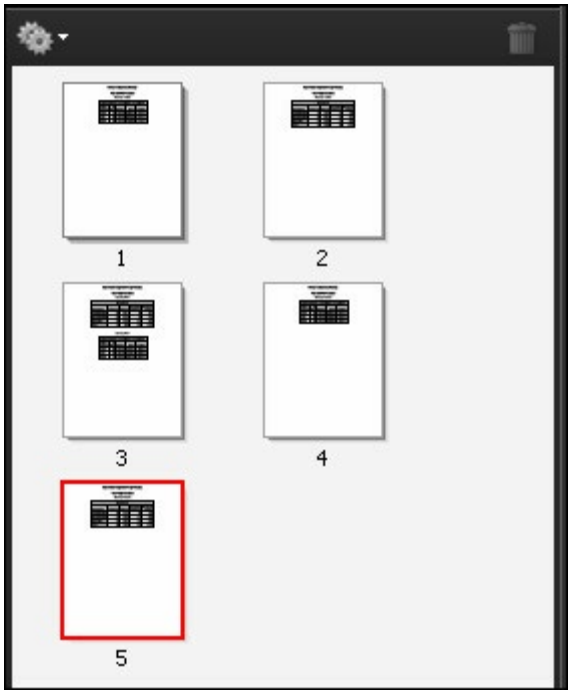
Under SAS 9.2, the Documents window does not give you the capability to interactively control pagination nor can you change the titles. However, PROC DOCUMENT has options that enable you to specify new titles, delete existing titles, insert page breaks, and remove page breaks.

The next example shows how to modify titles and page breaks in your document store. A later example shows how to create the same NEWHR document store using PROC DOCUMENT code.

Figure 11.13

Original Results on 4 Pages

WORK.NEWHR Results on 5 Pages



Changing Titles and Page Breaks

Before we remove the page breaks, we have to know the name of the output object that has the page break. A previous example demonstrates how to list the contents of a document store with the LIST statement in PROC DOCUMENT. Figure 11.14 shows the results of the LIST statement applied to document store WORK.NEWHR. This report lists the output objects and their internal names. Using the information from the LIST statement, the following PROC DOCUMENT step changes the titles and page breaks with the OBSTITLE and OBPAGE statements.

```
proc document name=work.newHR;
  obstitle \Central_America#1\Mexico#1\Summary#1;
  obstitle \US_Canada#1\Canada#1\Summary#1;
  obstitle \US_Canada#1\United_States#1\Summary#1;
  obpage \Central_America#1\Mexico#1\OneWayFreqs#1 /  delete;
  obpage \Central_America#1\Mexico#1\OneWayFreqs#1 /  after delete;
  obpage \US_Canada#1\Canada#1\OneWayFreqs#1  /  delete;
  obpage \US_Canada#1\United_States#1\Summary#1;
  obpage \US_Canada#1\United_States#1\OneWayFreqs#1 /  delete;
  obtitle1 \Central_America#1\Mexico#1\Summary#1
         'Salary and Work Force Report';
  obtitle1 \US_Canada#1\Canada#1\Summary#1
        'Salary and Work Force Report';
  obtitle1 \US_Canada#1\United_States#1\Summary#1
         'Salary and Work Force Report';
run;
quit;
```

The first three OBSTITLE statements delete “The Means Procedure” title.

The OBPAGE statements are specified three ways and they modify page breaks. Without any options, as shown in the fourth OBPAGE statement, OBPAGE inserts a page break. With the DELETE option, as shown in the first, third, and fifth statements, OBPAGE deletes the page break *before* an output object. With the AFTER DELETE option in the second statement, OBPAGE deletes the page break *after* the output object. This code deletes page breaks before the OneWayFreqs objects for all three countries, and it deletes the page break after the Mexico PROC FREQ output object as well. It also inserts a page break before the United States PROC MEANS output object.

Finally, the three OBTITLE1 statements change the title for each page break to the text enclosed in quotation marks.

Now, we need to replay the changed ODS document. Figure 11.15 shows on the left that the new output when sent to the PDF destination contains three pages. The first

page of the output is shown on the right.

```
ODS PDF FILE='NewHRDoc.pdf';
  proc document name=work.newhr;
    replay;
  run;
  quit;
ODS PDF CLOSE;
```

Figure 11.14

Output - (Untitled)

```
Listing of: \Work.Newhr\
Order by: Insertion
Number of levels: All
```

Obs	Path	Type
1	\Central_America#1	Dir
2	\Central_America#1\Mexico#1	Dir
3	\Central_America#1\Mexico#1\Summary#1	Table
4	\Central_America#1\Mexico#1\OneWayFreqs#1	Table
5	\US_Canada#1	Dir
6	\US_Canada#1\Canada#1	Dir
7	\US_Canada#1\Canada#1\Summary#1	Table
8	\US_Canada#1\Canada#1\OneWayFreqs#1	Table
9	\US_Canada#1\United_States#1	Dir
10	\US_Canada#1\United_States#1\Summary#1	Table
11	\US_Canada#1\United_States#1\OneWayFreqs#1	Table

Figure 11.15

Salary and Work Force Report

Country=Mexico

Analysis Variable : AnnualSalary Annual Salary				
Position Category	N Obs	Minimum	Mean	Maximum
Contract	15	35000.00	69200.00	140000.00
Hourly	32	35000.00	57390.63	150000.00
Salaried	46	35000.00	47565.22	114000.00

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Administration	12	12.90	12	12.90
Manufacturing	51	54.84	63	67.74
Marketing	17	18.28	80	86.02
R&D	13	13.98	93	100.00

Creating a New Document Store Using PROC DOCUMENT

Two examples before this one, we created and organized the document store NEWHR interactively. The hierarchy that you see in the SAS Results window is the same hierarchy that is originally created in the document store. What if you want a different organization or hierarchy for the objects created by SAS? For example, you might want to change the structure or hierarchy of the document objects that are produced by a nightly batch job. This example shows you how you can create and modify the organization of a document store using PROC DOCUMENT code.

The NAME= option in the PROC DOCUMENT statement creates a new document store. The MAKE statement creates folders and subfolders. The DIR statement navigates through the hierarchy of the document store. The SETLABEL statement names the folders.

The following PROC DOCUMENT step creates new document store NEWSALDOC, and it defines its folder hierarchy. The next example shows how PROC DOCUMENT can copy output objects into this new document store.

```
proc document name=work.newsaldoc(write);
  make Central_America, US_Canada;
run;
  dir;
  setlabel \Central_America#1 'Reports for Central America';
  setlabel \US_Canada#1 'Reports for United States and Canada';
run;
  dir Central_America;
  make Mexico;
  dir;
run;
  dir \US_Canada;
  make Canada;
  dir;
  dir \US_Canada;
  make United_States;
  setlabel \US_Canada #1\United_States#1 'United States';
  dir;
run;
quit;
```

The MAKE statement creates a new folder. Imagine there’s a pointer that knows where you are in the document store directory structure. The DIR command moves the pointer. Sometimes it is easiest to start at the top of the document store and then navigate down the structure. A DIR command with no location specification places you at the top or the root level of the directory structure. From the root directory, you can navigate down level by

level. Alternatively, you can find the directory name where you want to be and navigate down by specifying the entire path in the DIR statement.

For example, when PROC DOCUMENT opens a document store, the pointer is at the root level of the document store. The first MAKE statement in the preceding code creates two folders, Central_America and US_Canada, in the document store. The first DIR command goes back to the top of NEWSALDOC. Next, PROC DOCUMENT navigates to each of the two folders just created and uses the SETLABEL statement to name and describe the new folders. The remaining statements navigate through the hierarchy, make folders, and label folders until the NEWSALDOC document store looks like Figure 11.16. Sometimes it is easier to hold the document window open and submit each run group separately and watch the changes happen. Just a few examples of this process are shown in Figure 11.17.

Figure 11.16

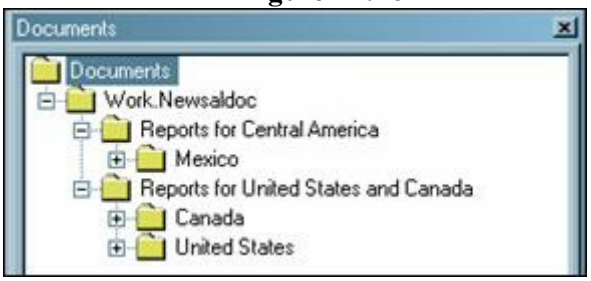
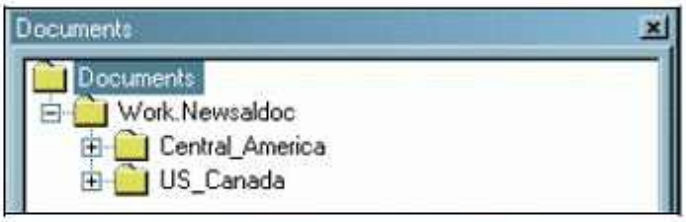


Figure 11.17

Run Group

```
proc document
      name=work.newsaldoc(write);
  make Central_America,
          US_Canada;
run;

  dir;
  setlabel \Central_America#1
   'Reports for Central America';
  setlabel \US_Canada#1
  'Reports for US and Canada';
run;
  dir Central_America;
  make Mexico;
  dir;
run;

  dir \US_Canada;
  make Canada;
  dir;
  dir \US_Canada;
  make United_States;
  setlabel
\US_Canada#1\United_States#1
            'United States';
  dir;
run;
quit;
```

Document Store After Each Run Group



Copying Objects into a New Document Store Using PROC DOCUMENT

After you have the hierarchy in the document store NEWSALDOC defined as shown in Figure 11.16, you can copy output objects from an existing ODS document to NEWSALDOC. For this example, we rerun the original PROC MEANS and PROC FREQ steps in the first example, but without SAS titles and without procedure titles, and then we save the document store in BASICSALDOC.

```
proc document name=work.newsaldoc(update);
  dir \work.newsaldoc\Central_America#1\Mexico#1;
  copy \work.basicsaldoc\Means#1\ByGroup2#1\Summary#1
    to \work.newsaldoc\Central_America#1\Mexico#1;
  copy \work.basicsaldoc\Freq#1\ByGroup2#1\Table1#1\OneWayFreqs#1
    to ^;
run;
  dir;
  dir \US_Canada#1;
  dir Canada;
  copy \work.basicsaldoc\Means#1\ByGroup1#1\Summary#1 to ^;
  copy \work.basicsaldoc\Freq#1\ByGroup1#1\Table1#1\OneWayFreqs#1 to ^;
run;
  dir;
  dir \US_Canada#1\United_States#1;
  copy \work.basicsaldoc\Means#1\ByGroup3#1\Summary#1 to ^;
  copy \work.basicsaldoc\Freq#1\ByGroup3#1\Table1#1\OneWayFreqs#1 to ^;
run;
quit;
```

The COPY statement might look familiar to you if you have used PC DOS or UNIX copy commands. Recall how in PC DOS you had to navigate the directory structure with CD (Change Directory) commands. You have to perform the same kind of navigation through the hierarchy in the ODS document store so that ODS copies the output object to the right location. There are a few different ways to navigate. For Mexico, the COPY statement is shown in both the long form in the first COPY statement and in the abbreviated form in the second COPY statement. The specification `to ^` means to copy to the current directory location. Using the explicit navigation path as in the first statement for Mexico might be easier to follow.

The key to PROC DOCUMENT syntax is using the correct output object names and understanding the directory path syntax. The directory pathname can come from the Properties window, as shown in the last example of Figure 11.17, or from the listing output of a PROC DOCUMENT step.

Figure 11.18 shows the structure of NEWSALDOC after the COPY statements have executed. Replaying the document to the PDF destination produces a five-page report, as shown in Figure 11.19.

Figure 11.18

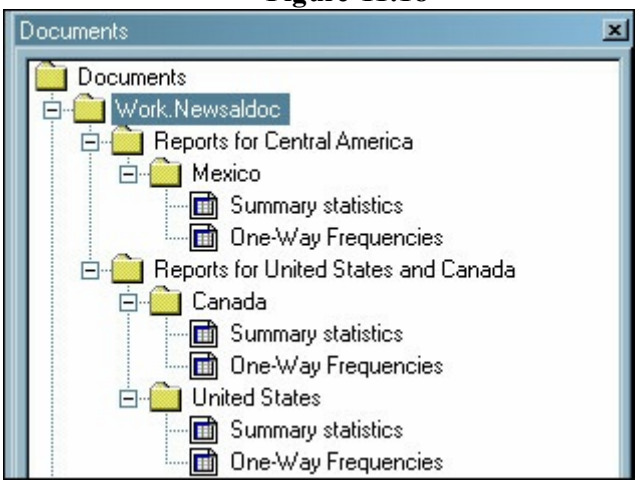
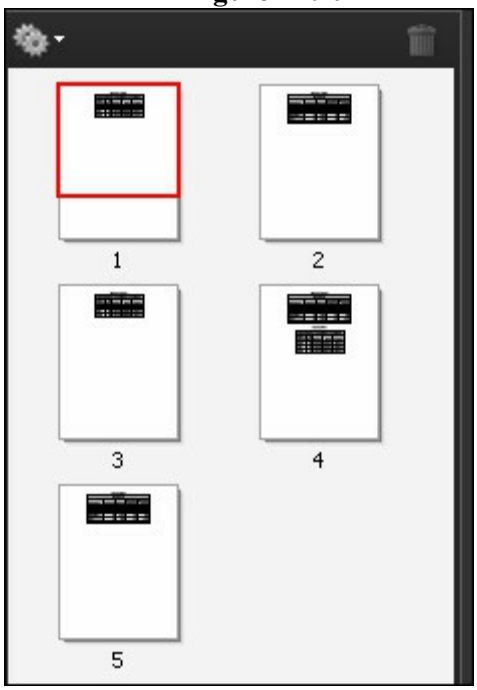


Figure 11.19



Adding Titles and Correcting Page Breaks after Copying Output Objects to a New Document

Taking a closer look at the PDF file in Figure 11.19 shows that only the BYLINE titles appear in the document and that some of the page breaks need to be removed. The following PROC DOCUMENT step removes page breaks, inserts appropriate page breaks, and inserts the desired titles.

```
proc document name=work.newsaldoc;
  obpage \Central_America#1\Mexico#1\Summary#1 /  after delete;
  obpage \Central_America#1\Mexico#1\OneWayFreqs#1 /  delete;
  obpage \US_Canada#1\Canada#1\Summary#1  /  after delete;
  obpage \US_Canada#1\Canada#1\OneWayFreqs#1  /  delete;
  obpage \US_Canada#1\Canada#1\Summary#1;
  obpage \US_Canada#1\United_States#1\Summary#1;
  obpage \US_Canada#1\United_States#1\Summary#1  /  after delete;
  obpage \US_Canada#1\United_States#1\OneWayFreqs#1 /  delete;
  obtitle1 \Central_America#1\Mexico#1\Summary#1
        'Salary and Work Force Report';
  obtitle1 \US_Canada#1\Canada#1\Summary#1
        'Salary and Work Force Report';
  obtitle1 \US_Canada#1\United_States#1\Summary#1
        'Salary and Work Force Report';
run;
quit;
```

After submitting this PROC DOCUMENT step, a replay to PDF produces a PDF file that has only three pages. Each BY group is on its own page and title text has been added. Review Figure 11.20 to see the newly rearranged document.

Whether you modify your document store interactively with the Documents window method or by submitting a PROC DOCUMENT step, you can rearrange your SAS output so it suits your needs without rerunning your analysis programs.

Figure 11.20

Page 1

Salary and Work Force Report

Country=Mexico

Analysis Variable : AnnualSalary Annual Salary				
Position Category	N Obs	Minimum	Mean	Maximum
Contract	15	35000.00	69200.00	140000.00
Hourly	32	35000.00	57390.63	150000.00
Salaried	46	35000.00	47565.22	114000.00

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Administration	12	12.90	12	12.90
Manufacturing	51	54.84	63	67.74
Marketing	17	18.28	80	86.02
R&D	13	13.98	93	100.00

Page 2

Salary and Work Force Report

Country=Canada

Analysis Variable : AnnualSalary Annual Salary				
Position Category	N Obs	Minimum	Mean	Maximum
Contract	25	35000.00	59940.00	105000.00
Hourly	52	35000.00	48605.77	115500.00
Salaried	96	35000.00	53552.08	143500.00

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Administration	29	16.76	29	16.76
Manufacturing	84	48.55	113	65.32
Marketing	34	19.65	147	84.97
R&D	26	15.03	173	100.00

Page 3

Salary and Work Force Report

Country=USA

Analysis Variable : AnnualSalary Annual Salary				
Position Category	N Obs	Minimum	Mean	Maximum
Contract	25	35000.00	43080.00	99500.00
Hourly	35	35000.00	47757.14	126500.00
Salaried	71	35000.00	51901.41	133500.00

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Administration	29	22.14	29	22.14
Manufacturing	68	51.91	97	74.05
Marketing	21	16.03	118	90.08
R&D	13	9.92	131	100.00

Using the SASEDOC LIBNAME Engine

The output objects created by ODS and stored in an ODS document store are usually just rows and columns of information—basically rectangular in structure. A special LIBNAME engine, SASEDOC, enables you to work with an ODS document store as if it were a data set.

The SASEDOC engine provides Read access to the document store, enabling you to treat the document store objects as data sets. You cannot write an output SAS data set to the library that points to an ODS document store. You can, however, rename and delete document stores with data set options. For example, you might want to simplify the object name for further reference by using PROC DATASETS.

When you look below at the PROC MEANS part of the WORK.ORIGDOC document store created in the first example, you can see that each BY group, which is highlighted, has its own directory folder.

Obs	Path	Type
1	\Means#1	Dir
2	\Means#1\ByGroup1#1	Dir
3	\Means#1\ByGroup1#1\Summary#1	Table
4	\Means#1\ByGroup2#1	Dir
5	\Means#1\ByGroup2#1\Summary#1	Table
6	\Means#1\ByGroup3#1	Dir
7	\Means#1\ByGroup3#1\Summary#1	Table

If you want to treat the table for Canada as a SAS data set, you first have to use the SASEDOC engine to specify its folder as a library:

```
LIBNAME cansum SASEDOC "\work.saldoc\Means\ByGroup1#1";
```

This syntax can be repeated for the other two BY groups. Figure 11.21 shows the SAS log for the three LIBNAME statements for the three BY groups. The BY groups are processed in alphabetical order by country name. Therefore, BY group 1 is Canada, BY group 2 is Mexico, and BY group 3 is USA.

After you've pointed to the folder in the document store, you now need to figure out the two-level name for use in SAS processes. "Summary#1" is not a valid SAS name because of the pound sign (#). The valid usage for the table name is CANSUM.SUMMARY, which is the libref and the object name without the number. Consider the following PROC CONTENTS, PROC PRINT, and PROC DATASETS steps. Output from these steps is shown in Figure 11.22.

```
proc contents data=cansum.summary;
run;
proc print data=cansum.summary;
 title 'Summary document object for Canada';
```

```
run;
proc datasets library=mexsum nolist;
    change summary=sumby2;
run;
quit;
```

If you have more than one SUMMARY item in the ODS document store, the new data set option DOC_SEQNO=, which can be used only with SASEDOC, enables you to reference an exact document sequence number. Refer to the documentation for the DOC_SEQNO= option.

Figure 11.21

```
548  LIBNAME cansum SASEDOC "\work.origdoc\Means\ByGroup1#1";
NOTE: Libref CANSUM was successfully assigned as follows:
      Engine:        SASEDOC
      Physical Name: \Work.Origdoc\Means#1\ByGroup1#1
549  LIBNAME mexsum SASEDOC "\work.origdoc\Means\ByGroup2#1";
NOTE: Libref MEXSUM was successfully assigned as follows:
      Engine:        SASEDOC
      Physical Name: \Work.Origdoc\Means#1\ByGroup2#1
550  LIBNAME usasum SASEDOC "\work.origdoc\Means\ByGroup3#1";
NOTE: Libref USASUM was successfully assigned as follows:
      Engine:        SASEDOC
      Physical Name: \Work.Origdoc\Means#1\ByGroup3#1
```

Figure 11.22

PROC CONTENTS Output (partial)

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
5	AnnualSalary_Max	Num	8	BEST12.	
4	AnnualSalary_Mean	Num	8	BEST12.	
3	AnnualSalary_Min	Num	8	BEST12.	
1	Category	Char	10		Position Category
2	NObs	Num	8	BEST3.	

PROC PRINT Output

Summary document object for Canada

Obs	Category	NObs	Annual Salary_Min	Annual Salary_Mean	Annual Salary_Max
1	Contract	25	35000	59940	105000
2	Hourly	52	35000	48605.769231	115500
3	Salaried	96	35000	53552.083333	143500

PROC DOCUMENT LIST Output (after PROC DATASETS)

Listing of: \Work.Origdoc\
Order by: Insertion
Number of levels: All

Obs	Path	Type
1	\Means#1	Dir
2	\Means#1\ByGroup1#1	Dir
3	\Means#1\ByGroup1#1\Summary#1	Table
4	\Means#1\ByGroup2#1	Dir
5	\Means#1\ByGroup2#1\Summary#1	Table
6	\Means#1\ByGroup3#1	Dir
7	\Means#1\ByGroup3#1\Summary#1	Table

Summary of PROC DOCUMENT Statements

Think of PROC DOCUMENT and ODS DOCUMENT as a team. Although there are some tasks, such as OBPAGE and OBTITLE statements, that you cannot do in the Documents window, between the Documents window and PROC DOCUMENT, the possibilities for document creation are virtually endless. For your ease of reference, we have listed the tasks that can be done only with PROC DOCUMENT.

Tasks for PROC DOCUMENT Only (SAS 9.1.3 and 9.2)	
Task	**PROC DOCUMENT Statement**
Create or delete a page break.	OBPAGE
Create, delete, or modify title lines.	OBTITLE
Create, delete, or modify subtitle lines.	OBSTITLE
Create or modify text before an output object.	OBBNOTE
Create or modify text after an output object.	OBANOTE
Create or modify footnote lines.	OBFOOTN
Create or update notes in the current folder.	NOTE
Import a data set or graph segment.	IMPORT

To learn the latest information about ODS DOCUMENT, refer to the SAS OnlineDoc.

Chapter 12
Using Tagsets to Generate MARKUP Output

ODS tagset templates were first introduced in SAS 8.2 as a way to enable users to control the markup language tags or markup text written to their result files.

The process of marking up page proofs or a manuscript is a manual process, generally performed by a writer or editor. In a computer file, the concept of markup has been extended to include formatting instructions inserted into a file or document text so that the appropriate rendering software can format the text in the document.

Generally, these formatting instructions take the form of tags or commands that start with a special character. For example, some commands start with a period (.), some start with a slash (/), some are enclosed in curly braces ({}), and the most frequently used markup languages, such as XML and HTML, use less than and greater than signs (< >) to delimit the tags or elements.

The "granddaddy" of all computer markup languages is SGML (Standard Generalized Markup Language). SGML is the foundation on which HTML (Hypertext Markup Language) and XML (Extensible Markup Language) are built. However, other markup languages exist.

ODS can produce almost any markup language you need. If there is not a predefined tagset template for your "flavor" of markup language, you can use the TEMPLATE procedure to create your own custom tagset. SAS creates markup language files in two ways: 1) through the SAS XML LIBNAME engine SXLE, which exports data from SAS data set format to XML format, or 2) with the ODS MARKUP family of destinations, which allows creation of various types of markup files from procedure or DATA step output. The scope of this chapter is limited to the creation of markup language result files from procedure or DATA step output using ODS and ODS MARKUP.

The data sets BILLINGS and PARTSAUCTION illustrate the examples in this chapter. These data sets are available on the author pages for this book at http://support.sas.com/authors.

ODS MARKUP Destination

You have many choices for destinations with the ODS MARKUP destination. The HTML destination is probably already familiar. To this you can add MSOFFICE2K, CSV, and EXCELXP. More obscure destinations include, but are not limited to, troff, LaTeX, DocBook, and Pyx.

The key to using ODS to create markup files is to understand what set of markup tags you need to generate. The method by which tagset templates are created for use with ODS is with PROC TEMPLATE. Tagset templates are stored in the SASHELP.TMPLMST template store. You can use PROC TEMPLATE to modify these tagsets or create your own custom tagset. We first describe how to use the ODS MARKUP destination with some tagset templates supplied by SAS.

Here are three ways to use the ODS MARKUP destination:

1. Use the one-level destination name. These one-level names, also called aliases, are set by the ODS developers. Examples of aliases are HTML, MSOFFICE2K, and CSV. Because the one-level destination names are set by the ODS developers, a limited number of tagset templates fall into this category.
2. Use ODS MARKUP with the TYPE= or TAGSET= option to define the MARKUP destination and use a particular tagset template.
3. Use ODS TAGSETS.<*tagsetname*> syntax to invoke the *tagset* template as an ODS destination.

The following program illustrates all three methods. All three ODS statements use the DOCBOOK XML tagset template. This tagset produces files in DOCBOOK format, which is used for writing technical papers and books. The three output files are all the same, regardless of invocation method. Method 1 works for the DOCBOOK tagset template invocation because it is one of the aliased tagsets. Method 2 and Method 3 should be used for non-aliased tagset templates, as discussed below.

```
ODS DOCBOOK FILE='docbook1.xml';
ODS MARKUP TAGSET=DOCBOOK FILE='docbook2.xml';
ODS TAGSETS.DOCBOOK FILE='docbook3.xml';
proc print data=billings;
  title 'Using Markup Tagsets';
  var Division JobType CustomerName BillableAmt;
run;
ODS _ALL_ CLOSE;
```

Figure 12.1 shows the first part of one of the output files (all three were the same). Note the XML prolog instruction at the beginning of the file and the less than (<) and greater than (>) brackets that delimit the tags. The <article> tag is the root tag in the document. These tags all conform to the DOCBOOK specification.

Not all tagset templates have aliases. The first part of Figure 12.2 shows the contents of

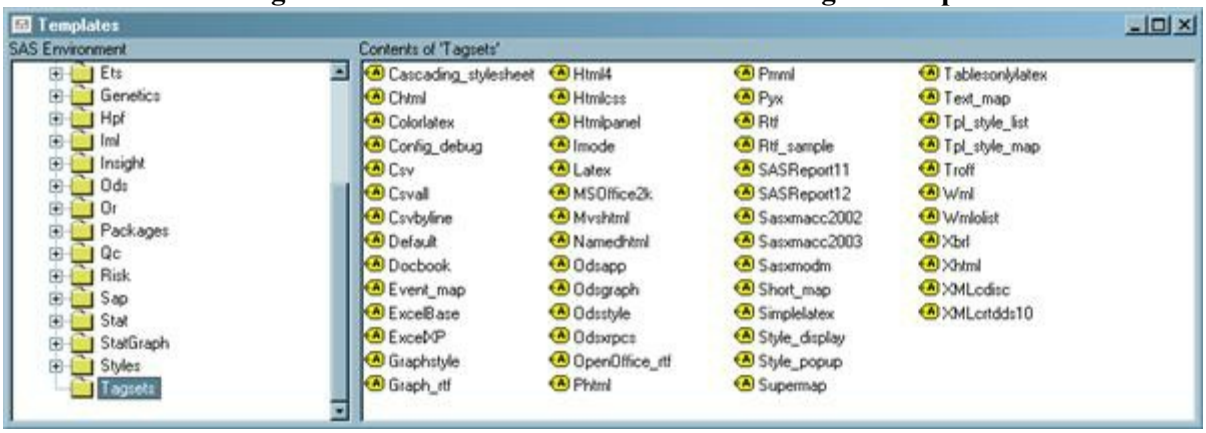
the TAGSETS folder in the SASHELP.TMPLMST template store. This folder contains both aliased and non-aliased tagset template definitions. The table underneath Figure 12.2 lists the destination aliases in SAS 9.2. Any tagset template that does not appear in this table must be invoked with Method 2 or Method 3.

Most of the examples in this chapter use the third style of syntax: ODS TAGSETS. *<tagsetname>*.

Figure 12.1

```
<?xml version="1.0" encoding="windows-1252"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
                 "http://www.docbook.org/docbook/xml/4.3/docbookx.dtd">

<article>
<articleinfo>
    <pubdate>2008-12-18 17:54:49</pubdate>
    <productname>SAS</productname>
    <productnumber>9.2</productnumber>
</articleinfo>
<section>
<title>Using Markup Tagsets</title>
<section>
<title>Print</title>
<section>
<title>Data Set WORK.BILLINGS</title>
<para>
<informaltable colsep="1" rowsep="1" frame="all">
<tgroup cols="5">
<colspec colname="col1" colwidth="8*"/>
<colspec colname="col2" colwidth="13*"/>
<colspec colname="col3" colwidth="10*"/>
<colspec colname="col4" colwidth="23*"/>
<colspec colname="col5" colwidth="8*"/>
<thead>
<row>
<entry>Obs</entry>
<entry>Division</entry>
<entry>JobType</entry>
<entry>CustomerName</entry>
<entry>BillableAmt</entry>
</row>
```

Figure 12.2: All SAS 9.2 ODS MARKUP Tagset Templates



List of ODS Aliases for MARKUP Tagset Templates in SAS 9.2

CHTML	CSVALL	HTML	HTMLCSS	LATEX	PHTML	WML
CSV	DOCBOOK	HTML4	IMODE	MSOFFICE2K	TROFF	WMLOLIST

Creating CSV Files with ODS MARKUP

A CSV file contains columns of data values that are separated by commas. It is a common way to transfer data in and out of Microsoft Excel and other software programs.

Three SAS tagset templates can create CSV files:

1. ODS CSV creates a CSV output file that contains only the column headers and data values.
2. ODS CSVALL creates a CSV output file that contains the column headers, data values, titles, BY lines, and notes.
3. ODS TAGSETS.CSVBYLINE creates a CSV output file that contains the column headers, data values, and BY lines.

The following code invokes the three tagset templates. As you might imagine from their names, there are differences in the data contained in each of the three outputs.

```
ODS CSV FILE='result1.csv';
ODS CSVALL FILE='result2.csv';
ODS TAGSETS.CSVBYLINE FILE='result3.csv';
proc print data=billings;
  by Division JobType;
  title 'Using CSV Tagsets';
  var JobType CustomerName BillableAmt;
run;
ODS _ALL_ CLOSE;
```

Figure 12.3 shows the CSV output. Figure 12.4 shows the CSVALL output. Figure 12.5 shows the CSVBYLINE output. The three outputs are shown first in Microsoft Notepad and then in Microsoft Excel.

To display in the SAS log the options and usage information available with the CSV tagset, you can add the DOC= 'HELP' suboption to your ODS CSV statements as follows.

```
ODS CSV FILE='result1.csv' options(Doc='Help');
ODS CSVALL FILE='result2. csv' options(Doc='Help');
ODS TAGSETS.CSVBYLINE FILE='result3.csv' options(Doc='Help');
```

When SAS adds new suboptions to any of the tagset templates that have internal documentation, such as the CSV tagsets, you can display the updated information by adding the DOC= 'HELP'
suboption to your ODS statement. At the end of the chapter, we've included the text of the CSV internal documentation.

Another useful ODS CSV suboption is DELIMITER=. This suboption enables you to specify the delimiter that separates your columns of data values. For example, if you wanted to use a colon instead of a comma, you would specify:

```
ODS CSV FILE='result1b.csv' options(delimiter=':');
```

Figure 12.3

```
"Obs","JobType","CustomerName","BillableAmt"
"1","Consulting","Bricks and Mortar, Inc.","$621.83"
"2","Consulting","Sweat Shops Athletic","$124.79"
"3","Consulting","Bricks and Mortar, Inc.","$574.79"
"4","Consulting","Virtual Co.","$980.79"
"5","Consulting","IPO.com","$484.80"
```

	A	B	C	D
1	Obs	JobType	CustomerName	BillableAmt
2	1	Consulting	Bricks and Mortar, Inc.	$621.83
3	2	Consulting	Sweat Shops Athletic	$124.79
4	3	Consulting	Bricks and Mortar, Inc.	$574.79
5	4	Consulting	Virtual Co.	$980.79
6	5	Consulting	IPO.com	$484.80

Figure 12.4

```
Using CSV Tagsets

"Division=Analysis Job Type=Consulting"
"Obs","JobType","CustomerName","BillableAmt"
"1","Consulting","Bricks and Mortar, Inc.","$621.83"
"2","Consulting","Sweat Shops Athletic","$124.79"
"3","Consulting","Bricks and Mortar, Inc.","$574.79"
"4","Consulting","Virtual Co.","$980.79"
"5","Consulting","IPO.com","$484.80"
```

	A	B	C	D
1	Using CSV Tagsets			
2				
3	Division=Analysis Job Type=Consulting			
4	Obs	JobType	CustomerName	BillableAm
5	1	Consulting	Bricks and Mortar, Inc.	$621.83
6	2	Consulting	Sweat Shops Athletic	$124.79
7	3	Consulting	Bricks and Mortar, Inc.	$574.79
8	4	Consulting	Virtual Co.	$980.79
9	5	Consulting	IPO.com	$484.80

Figure 12.5

```
"Division=Analysis Job Type=Consulting"
"Obs","JobType","CustomerName","BillableAmt"
"1","Consulting","Bricks and Mortar, Inc.","$621.83"
"2","Consulting","Sweat Shops Athletic","$124.79"
"3","Consulting","Bricks and Mortar, Inc.","$574.79"
"4","Consulting","Virtual Co.","$980.79"
"5","Consulting","IPO.com","$484.80"
```

	A	B	C	D
1	Division=Analysis Job Type=Consulting			
2	Obs	JobType	CustomerName	BillableAmt
3	1	Consulting	Bricks and Mortar, Inc.	$621.83
4	2	Consulting	Sweat Shops Athletic	$124.79
5	3	Consulting	Bricks and Mortar, Inc.	$574.79
6	4	Consulting	Virtual Co.	$980.79
7	5	Consulting	IPO.com	$484.80

Creating HTML Result Files with ODS MARKUP

Chapter 3 shows that you can create many flavors of HTML output files with ODS. Table 12.1 summarizes the different HTML choices you can specify using ODS. Since SAS 8.2, all of the HTML destinations (except ODS HTML3) are ODS MARKUP destinations, and they all use tagset templates to create output.

Table 12.1: Summary of Different HTML Formats Available with ODS

Method for HTML File Creation	SAS®9 Behavior
ODS HTML ... ;	Produces HTML 4.01 tags with an inline <STYLE> section.
ODS HTML3 ...;	Produces HTML 3.2 tags with embedded <FONT> tags.
ODS TAGSETS.XHTML ...;	Produces XHTML 1.0 tags in an HTML file with an inline <STYLE> section.
ODS MSOFFICE2K ...;	Creates an HTML file with tags and style attributes according to the Microsoft Office specification.
ODS HTMLCSS ...;	Produces HTML 3.2 tags that are more compatible with the Netscape Navigator browser.
ODS PHTML ...;	Produces HTML 3.2 compliant tags that use a minimal number of style elements for formatting (known as Plain HTML).
ODS CHTML ...;	Creates an HTML file with no style elements for formatting (known as Compact HTML).

The most notable difference is between CHTML output and output from the other HTML destinations. The following program creates an HTML, a CHTML, and a PHMTL file.

```
ODS HTML FILE='html4.html' STYLE=sasweb;
ODS PHTML FILE='plain.html' STYLE=sasweb;
ODS CHTML FILE='compact.html' STYLE=sasweb;
proc print data=billings;
  title 'Using Markup Tagsets';
  var Division JobType CustomerName BillableAmt;
run;
ODS _ALL_ CLOSE;
```

Although each ODS statement above specifies the STYLE=Sasweb option, the style specifications are not used the same way in the output files The major difference between the HTML file and the PHTML file is that the PHTML file has a limited number of CSS style properties, whereas the regular HTML file has more style properties. However, in this example, as seen in Figure 12.6, both files look essentially the same. The CHTML output shown in Figure 12.7 does not use any Sasweb style properties or have an inline style section.

Figure 12.6

Using Markup Tagsets

Obs	Division	JobType	CustomerName	BillableAmt
1	Analysis	Consulting	Bricks and Mortar, Inc.	$621.83
2	Analysis	Consulting	Sweat Shops Athletic	$124.79
3	Analysis	Consulting	Bricks and Mortar, Inc.	$574.79
4	Analysis	Consulting	Virtual Co.	$980.79
5	Analysis	Consulting	IPO.com	$484.80

Using Markup Tagsets

Obs	Division	JobType	CustomerName	BillableAmt
1	Analysis	Consulting	Bricks and Mortar, Inc.	$621.83
2	Analysis	Consulting	Sweat Shops Athletic	$124.79
3	Analysis	Consulting	Bricks and Mortar, Inc.	$574.79
4	Analysis	Consulting	Virtual Co.	$980.79
5	Analysis	Consulting	IPO.com	$484.80

Figure 12.7

Using Markup Tagsets

Obs	Division	JobType	CustomerName	BillableAmt
1	Analysis	Consulting	Bricks and Mortar, Inc.	$621.83
2	Analysis	Consulting	Sweat Shops Athletic	$124.79
3	Analysis	Consulting	Bricks and Mortar, Inc.	$574.79
4	Analysis	Consulting	Virtual Co.	$980.79
5	Analysis	Consulting	IPO.com	$484.80

Creating XML Files with ODS MARKUP

ODS includes tagset templates that can create several different kinds of predefined XML output. Table 12.2 describes several of the XML tagsets included in SAS®9.

Table 12.2: Selected XML Tagsets Available with ODS and SAS®9

Tagset Name	General Results of Tagset in SAS®9
DEFAULT	Produces XML tags designed specifically for SAS output files created with ODS.
DOCBOOK	Creates XML that conforms to the DOCBOOK specification as outlined at http://www.docbook.org/specs/.
EXCELXP	Creates a SpreadsheetML XML file with tags and style attributes according to the Microsoft Office 2002/2003 specification.
PMML	Creates a Predictive Model Markup Language file with tags according to the PMML 2.1 specification. See http://www.dmg.org/v2-1/GeneralStructure.html.
WML WMLOLIST	Creates a Wireless Markup Language file with tags according to the specification outlined at http://www.wapforum.org/what/technical.htm.
XBRL	Creates an Extensible Business Reporting Language XML file from **two** files: a header file of metadata and a data file. For more information about this tagset, see the SAS support Web site.

The XML tagset that is probably of most interest to spreadsheet users is the EXCELXP tagset, which produces a SpreadsheetML file, which conforms to the Microsoft specification for describing a spreadsheet workbook with XML tags. This XML describes almost every spreadsheet customization, such as formulas, data, and formatting. The following code produces two different XML files. The first ODS statement produces marked up PROC PRINT output using the DEFAULT tagset. The second ODS statement marks up the output using the EXCELXP tagset.

```
ODS MARKUP TAGSET=DEFAULT FILE='DefaultTagset.xml' STYLE=sasweb;

ODS TAGSETS.EXCELXP FILE='ExcelXPTagset.xml' STYLE=sasweb;

proc print data=billings;

  title 'Using Markup Tagsets';

  var Division JobType CustomerName BillableAmt;

run;

ODS _ALL_ CLOSE;
```

Figure 12.8 presents the first few lines of the DefaultTagset.xml file when viewed with Microsoft WordPad. Compare this code with the WordPad and Excel views of the ExcelXPTagset.xml file as shown in Figure 12.9. Note that the EXCELXP markup information and data are more complex and include instructions to create an Excel worksheet. Although both files are XML files, the EXCELXP XML can be rendered by Excel. The DefaultTagset.xml file would look slightly different in Internet Explorer than in WordPad, but it would still be an ASCII XML text file.

We've used PROC PRINT with the WORK.BILLINGS data for this and all the previous

examples. The best way to get to know each of the available HTML and XML tagsets is to create this sample output for each one and then view the results in a text editor. It's all basically the same output, but with different markup information added depending on the tagset template.

When creating an XML file of your SAS results, you must select the XML tagset most appropriate for the output. For example, an archive application could require different XML tags than a tracking application. Both applications could process XML output from ODS if the output was marked up with the appropriate XML tags. You will see examples of this in upcoming topics.

Figure 12.8

```
<?xml version="1.0" encoding="utf-8"?>

<odsxml>
<head>
<meta operator="username"/>
</head>
<body>
<proc name="Print">
<label name="IDX"/>
<title class="SystemTitle" toc-level="1">Using Markup Tagsets</title>
<branch name="Print" label="The Print Procedure" class="ContentProcName" toc-level="1">
<leaf name="Print" label="Data Set WORK.BILLINGS" class="ContentItem" toc-level="2">
<output name="Print" label="Data Set WORK.BILLINGS" clabel="Data Set WORK.BILLINGS">
<output-object type="table" class="Table">

  <style>
    <border spacing="1" padding="7" rules="none" frame="box"/>
  </style>
```

Figure 12.9

```
<?xml version="1.0" encoding="windows-1252"?>

<?mso-application progid="Excel.Sheet"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
          xmlns:x="urn:schemas-microsoft-com:office:excel"
          xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
          xmlns:html="http://www.w3.org/TR/REC-html40">
<DocumentProperties xmlns="urn:schemas-microsoft-com:office">
<Author>username</Author>
<LastAuthor>username</LastAuthor>
<Created>2009-02-23T17:03:26</Created>
<LastSaved>2009-02-23T17:03:26</LastSaved>
<Company>SAS Institute Inc. http://www.sas.com</Company>
<Version>9.02.01M0P02052008</Version>
</DocumentProperties>
<Styles>
<Style ss:ID="_body">
 <Interior ss:Pattern="Solid" />
 <Protection ss:Protected="1" />
</Style>
```

	A	B	C	D	E
1	Obs	Division	JobType	CustomerName	BillableAmt
2	1	QA	Consulting	Sweat Shops Athletic	$388.79
3	2	Documentation	Consulting	Smith & Smith	$929.81
4	3	Applications	In-house	Smith & Smith	$354.81
5	4	Reporting	Consulting	Smith & Smith	$799.80
6	5	Systems	Consulting	Virtual Co.	$661.80
7	6	Reporting	Consulting	IPO.com	$439.85

Learning More about ODS MARKUP Tagsets

Before you can create a custom tagset template or modify an existing tagset template, you need to understand how tagset templates work. The ODS MARKUP family of destinations includes a group of tagset templates known as diagnostic tagsets. You can use these tagset templates to reveal information about the ODS process of creating your output files.

The actions of the diagnostic tagset templates can be grouped into two categories: examination of ODS table and style elements and attributes, and examination of SAS events used with the tagset templates. Table 12.3 shows the diagnostic tagsets and the type of information they reveal.

Table 12.3: Diagnostic Tagsets Available with SAS®9 ODS

Tagset Name	Type of File Created	Type of Information Revealed	Tagset Name	Type of File Created	Type of Information Revealed
STYLE_DISPLAY	HTML	Style	TPL_STYLE_MAP	XML	Style
STYLE_POPUP	HTML	Style	EVENT_MAP	XML	Event
ODSSTYLE	SAS/Text	Style inheritance	SHORT_MAP	XML	Event
NAMEDHTML	HTML	Table and Style	TEXT_MAP	TXT	Event
TPL_STYLE_LIST	HTML	Style			

The following code uses the STYLE_POPUP tagset template. It creates an HTML file in which you can click on elements of the report to reveal the name of the element and associated style information. Figure 12.10 shows the popup window after a data cell has been selected.

```
ODS TAGSETS.STYLE_POPUP FILE='StyleDiag.HTML';
proc print data=billings(obs=5);
run;
ODS _ALL_ CLOSE;
```

You can see from Figure 12.10 that the STYLE_POPUP output reveals style element information. Replacing STYLE_POPUP with NAMEDHTML in the preceding ODS statement shows the kind of table information that you can obtain with the NAMEDHTML tagset. The NAMEDHTML output in Figure 12.11 displays the output object name in the table template. Output from the STYLE_POPUP tagset tells you the name of the style element to change and output from the NAMEDHTML tagset tells you the name of the table template item to change.

The XML diagnostic tagsets display very different results. Output using the SHORT_MAP tagset shows which events are triggered by the procedure and the order in which the events were triggered. Figure 12.12 shows some of the XML when applying the SHORT_MAP tagset to the preceding PROC PRINT step.

Once you understand the output being produced and the markup language tags required by

your application, you need to understand how ODS can produce the kind of markup language file you need. The key to this understanding is how events work. ODS sends the output to the destination using event processing. For example, every discrete piece of output processing that ODS must perform could be considered an event: the DOC event signals the beginning of result file creation, the SYSTEM_TITLE event carries SAS title information, the HEADER event carries column header information, and the DATA event carries data values.

Figure 12.10

Figure 12.11

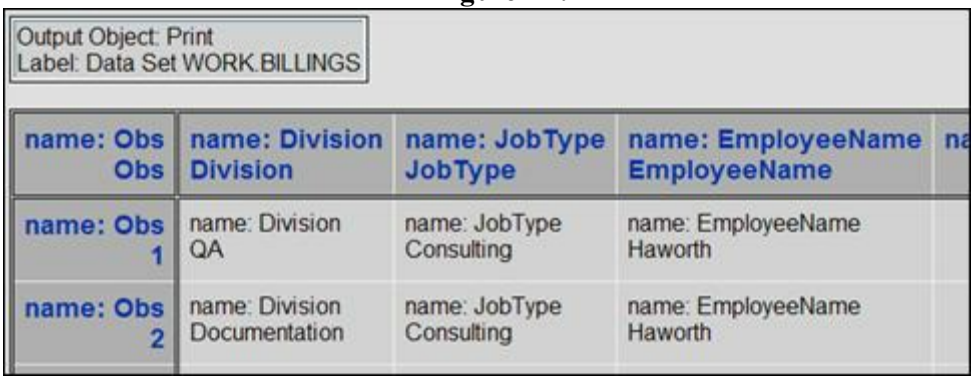


Figure 12.12

```
<?xml version="1.0" encoding="windows-1252"?>

<doc operator="username" sasversion="9.2" saslongversion="9.02.01M0P02052008" date="2009-2-24"
        time="13:50:09" encoding="windows-1252">
  <doc_head>
    <doc_meta/>
    <auth_oper/>
    <doc_title/>
    <stylesheet_link/>
    <javascript>
      <startup_function>
      </startup_function>
      <shutdown_function>
      </shutdown_function>
    </javascript>
  </doc_head>
  <doc_body>
    <proc name="Print">
      <anchor name="IDX"/>
      <page_setup>
      </page_setup>
```

Modifying Tagset Events

This example shows how to create your own tagset template and modify how the tagset handles an event. The new tagset is needed for a reports archive application. Assume that an archive application reads XML files on a Web site and looks for an <archive> tag, which delimits an archival date in the XML file. When users request the XML file from a Web site, the archive application compares the value of the <archive> tag and the current date, and only serves files with archival dates later than the current date. The Web site application is not the focus here. What we must examine is the XML needed by the Web site application.

The tagset REPARCHIVE, which this example defines, is based on the original DOCBOOK tagset template. The DOCBOOK tagset produced the output in Figure 12.1. The single difference from its usage in Figure 12.1 and its usage in this example is that instead of an <articleinfo> section, the output file must contain the <archive> tag.

Notice the <articleinfo> section near the top of Figure 12.1. The DOC_HEAD event produces the <articleinfo> tag. Here is the code for the DOC_HEAD event in the DOCBOOK tagset:

```
define event doc_head;
  start: put "<articleinfo>" NL;
    put "  <title>" BODY_TITLE "</title>" NL /
                if exists( BODY_TITLE);
    put "  <author>" AUTHOR "</author>" NL /if exists( AUTHOR);
    put "  <pubdate>" DATE " " TIME "</pubdate>" NL /
                if exists( DATE) and exists ( TIME);
    put "  <productname>SAS</productname>" NL;
    put "  <productnumber>" SASVERSION "</productnumber>" NL /
                if exists( SASVERSION);
  finish: put "</articleinfo>" NL;
end;
```

To create the REPARCHIVE tagset for the archive application, all you need to do is modify the DOC_HEAD event. The PARENT statement allows all of the original DOCBOOK tags to be created as in the original output, except for the changed DOC_HEAD section. The archive date is passed to the tagset via the SAS macro variable ARCDATE. The MVAR statement inside PROC TEMPLATE declares the macro variable ARCDATE. A %LET statement assigns a value to ARCDATE before executing the report.

When ODS applies the REPARCHIVE tagset to XML output from a SAS program, it inserts the archive date in the output file and does not insert an <articleinfo> section.

```
ODS PATH work.tagtemp(update) sashelp.tmplmst(read);
proc template;
  define tagset tagsets.reparchive / store=work.tagtemp;
    notes "Document Archive XML";
    parent = tagsets.docbook;
    mvar arcdate;
    define event doc_head;
      start: put "<archive>";
             put arcdate;
             break;
      finish: put "</archive>" NL;
              break;
    end;
  end;
run;
%let arcdate = 2008-12-31;
ODS TAGSETS.REPARCHIVE FILE='BillingsReport.xml';
proc print data=billings(obs=5);
run;
ODS _ALL_ CLOSE;
```

The STORE= option places the tagset in a temporary store (based on the ODS PATH statement). The PARENT= attribute instructs ODS to use the DOCBOOK tagset as the parent tagset template. PROC TEMPLATE replaces the DOC_HEAD event from DOCBOOK with the new instructions specified in the DEFINE EVENT DOC_HEAD block. Note that macro variable references inside a template definition are not prefaced with a leading ampersand (&). The output from this tagset template is shown in Figure 12.13.

As shown above, tagset template event code is mostly composed of PUT statements. Refer to SAS documentation for more detail on the PROC TEMPLATE statements that redefine the DOC_HEAD event.

Figure 12.13

```
<?xml version="1.0" encoding="windows-1252"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
                  "http://www.docbook.org/docbook/xml/4.3/docbookx.dtd">

<article>
<archive>2008-12-31</archive>
<section>
<section>
<title>Print</title>
<section>
<title>Data Set WORK.BILLINGS</title>
```

Creating a Custom Tagset

This example creates a custom XML tagset for an application that tracks the activity of an online auction of automobile parts. The only columns in the file with activity information are the part number, the part description, the seller's ID, and the asking price. There is one row for each part number. Assume that other files on the auction Web site contain contact information and purchaser information. The contents of the file are as follows:

partnum	partdesc	sellerid	askprice
123456	4 Truck Tires	TYW23918	100
121212	Sedan Shop Manual	JHC73619	15
987654	Minivan Rearview Mirror	UEM17640	55
989898	SUV Foglights	BND32080	125

The PROC TEMPLATE step that creates the new tagset PARTSAUCTION is on the facing page. Notice that it does not have a PARENT= statement as the previous example did. The entirety of the definition of the PARTSAUCTION tagset is completed in the PROC TEMPLATE step.

The root tag for the XML file is the <parts> tag, and it is specified in the DEFINE block for the DOC event. The code includes a separate section for the column headers, which is specified in the DEFINE block for the HEADER event. The DEFINE block for the ROW event specifies a <row> tag for every row. The DEFINE block for the DATA event specifies a <data> tag for every data cell.

The only event in the above tagset that is not tied directly to the data is the DOC event. Think of the DOC event as being linked to what happens when an operating system file is created or opened for writing, and then closed when your job is over. First, ODS needs to start the file, so the DOC event has a start section. Then, the other events in the tagset template cause procedure data and markup tags to be written to the file. Then, ODS needs to close the file so that the finish section of the DOC event will be executed. The DOC event is used only twice: when the operating system file is opened and when it is closed.

The tagset PARTSAUCTION specifies several events, and these events produce XML output that conforms to the hypothetical requirements of the online tracking application. What would happen if we used the PARTSAUCTION tagset with a PROC PRINT of the WORK.BILLINGS data? Although we can use the PARTSAUCTION tagset template with the WORK.BILLINGS data set, the chances are good that our parts auction XML application would not process the BILLINGS data set correctly. However, we will explore that possibility in the next section.

```
ODS PATH work.tagtemp(update) sashelp.tmplmst(read);

proc template;

  define tagset tagsets.partsauction / store=work.tagtemp;
```

```
define event doc;
  start: put "<?xml version=""1.0""";
         putq " encoding=" encoding / if exists( $show_charset);
         put "?>" NL NL;
         put "<parts>" NL;
         ndent;
         break;
  finish: xdent;
          put "</parts>" NL;
          break;
end;
define event header;
  start: ndent;
         put "<header>";
         put VALUE;
         break;
  finish: xdent;
          put "</header>" NL;
          break;
end;
define event row;
  start: ndent;
         put "<row>" NL;
         break;
  finish: xdent;
          put "</row>" NL;
          break;
end;
```

```
    define event data;
      start: ndent;
            put "<data>";
            put VALUE;
            break;
      finish: xdent;
             put "</data>" NL;
             break;
    end;
    output_type = "xml"; nobreakspace = " ";
    mapsub = %nrstr("/&lt;/&gt;/&amp;/");
    map = %nrstr("<>&");
  end;
run;
```

Using a Custom Tagset Template

The following code uses the tagset template PARTSAUCTION created in the previous example to create the XML output file PARTSLIST. Figure 12.14 shows the XML file created using PROC PRINT on the WORK.PARTSAUCTION data set.

```
ODS PATH work.tagtemp(update) sashelp.tmplmst(read);
ODS TAGSETS.PARTSAUCTION FILE='PartsList.xml';
proc print data=partsauction noobs;
  title 'Online Parts Auction';
  var partnum partdesc sellerid askprice;
run;
ODS _ALL_ CLOSE;
```

Comparing the final XML output file in Figure 12.14 to the PROC TEMPLATE code that created the output file, you can see that the row events and the data events executed more than once. The header event executed once for each column header. The DOC event executed twice in the event stream: once at the beginning and once at the end as indicated by the <parts> and </parts> tags. Even though the PROC PRINT step has a TITLE statement, there is no evidence of the TITLE statement being written to the XML output file, because there was no event block to handle or write XML tags around the value of the TITLE statement.

The XML output shows how a single event can execute more than once and how the event causes the tagset template to write output based on the order of the event in the event stream, and *not* on the order of the event in the PROC TEMPLATE tagset definition.

If we use the PARTSAUCTION tagset template with the BILLINGS data set, we can see in Figure 12.15 that ODS inserts the same markup tags around the BILLINGS data. The following program applies the PARTSAUCTION template to the PROC PRINT report of data set BILLINGS:

```
ODS PATH work.tagtemp(update) sashelp.tmplmst(read);
ODS TAGSETS.PARTSAUCTION FILE='BillingParts.xml';
proc print data=billings(obs=2) noobs;
  var division employeename customername hours jobtype;

run;
ODS _ALL_ CLOSE;
```

Of course, the BILLINGS data is probably not acceptable to the Web site auction application, but this program does illustrate that the tagset handles events the same way regardless of the data.

Event processing and tagset creation are powerful features of SAS and ODS. This chapter touches on the basics of tagset creation. For more detail on developing your skills in tagset development, refer to the SAS documentation.

Figure 12.14

```xml
<?xml version="1.0"?>

<parts>
<row><header>partnum</header>
<header>partdesc</header>
<header>sellerid</header>
<header>askprice</header>
</row>
<row><data>123456</data>
<data>4 Truck Tires</data>
<data>TYW23918</data>
<data>100</data>
</row>
<row><data>121212</data>
<data>Sedan Shop Manual</data>
<data>JHC73619</data>
<data> 15</data>
</row>
<row><data>987654</data>
<data>Minivan Rearview Mirror</data>
<data>UEM17640</data>
<data> 55</data>
</row>
<row><data>989898</data>
<data>SUV Foglights</data>
<data>BND32080</data>
<data>125</data>
</row>
</parts>
```


Figure 12.15


```xml
<?xml version="1.0"?>

<parts>
<row><header>Division</header>
<header>EmployeeName</header>
<header>CustomerName</header>
<header>Hours</header>
<header>JobType</header>
</row>
<row><data>QA</data>
<data>Haworth</data>
<data>Sweat Shops Athletic</data>
<data>     4.0</data>
<data>Consulting</data>
</row>
<row><data>Documentation</data>
<data>Haworth</data>
<data>Smith & Smith</data>
<data>    10.0</data>
<data>Consulting</data>
</row>
</parts>
```

More about Tagset Templates

One of the good things about ODS tagset templates is that you can download updated versions of the tagset templates without waiting for a new release of SAS. A call to SAS Technical Support with a tagset template question is likely to be answered one of two ways: "Check whether you've downloaded the most current version of the tagset definition," or "The problem you've encountered is fixed in the most current version of the tagset definition."

This means that some housekeeping is involved in using tagset templates with ODS. Over time, users have asked the ODS developers for fixes and enhancements to the tagset templates used with ODS MARKUP. The updated code is posted on the SAS Web site. If you like the output produced by the currently installed tagset templates, then you have no housekeeping tasks. However, if the phrase "updated tagset template" has ever arisen, then you will find this topic useful.

The ODS developers maintain a Web site for updates to ODS MARKUP tagset templates. Check the SAS Web site for links to ODS tagset updates. Figure 12.16 shows a view of the Web site as it appeared at the writing of this book.

Figure 12.17 shows the text from the middle of the ODS MARKUP Web page entitled "Just the tagsets." The links connect you to ZIP archives of updated tagset templates for your SAS version. Note that a tagset template definition designed for SAS®9 and later will probably not work in earlier versions. Enhancements included in SAS®9 tagset definitions are not likely to work in SAS 8.

Continuing down this same Web page, another section, "Updated SAS 9.1 Tagsets to Download Individually," has links to specific tagset updates. Figure 12.18 shows a partial screenshot of this section that starts with links to update CSV and ExcelXP tagsets.

These two sets of tagsets are probably the most frequently used ODS MARKUP tagset templates, followed by updates to the HTML tagset templates. Clicking a link in this section takes you to Web page that contains an ASCII text file. For example, selecting the most current ExcelXP tagset template connects you to the text file EXCLTAGS.TPL. You need to save this code in its entirety to a location on your disk or server. Then you need to use SAS to execute the code to create an updated copy of the tagset template in the first template store that you can write to, such as SASUSER.TEMPLAT. You could also use an ODS PATH statement as discussed in previous examples to establish a path to a permanent template store in another user-defined library.

Do not worry about the file extension of .TPL; it's actually a good thing. Unless you've associated this file extension with SAS, the only thing that should happen when you double-click a TPL file after you've saved it is that it should open in Notepad or a text editor.

Similarly, if you clicked the link shown in Figure 12.18 for the CSV tags, you would be directed to a file called CSVTAGS.TPL. As with the ExcelXP tagset, you only need to save this file to your system and then execute the code.

Figure 12.16

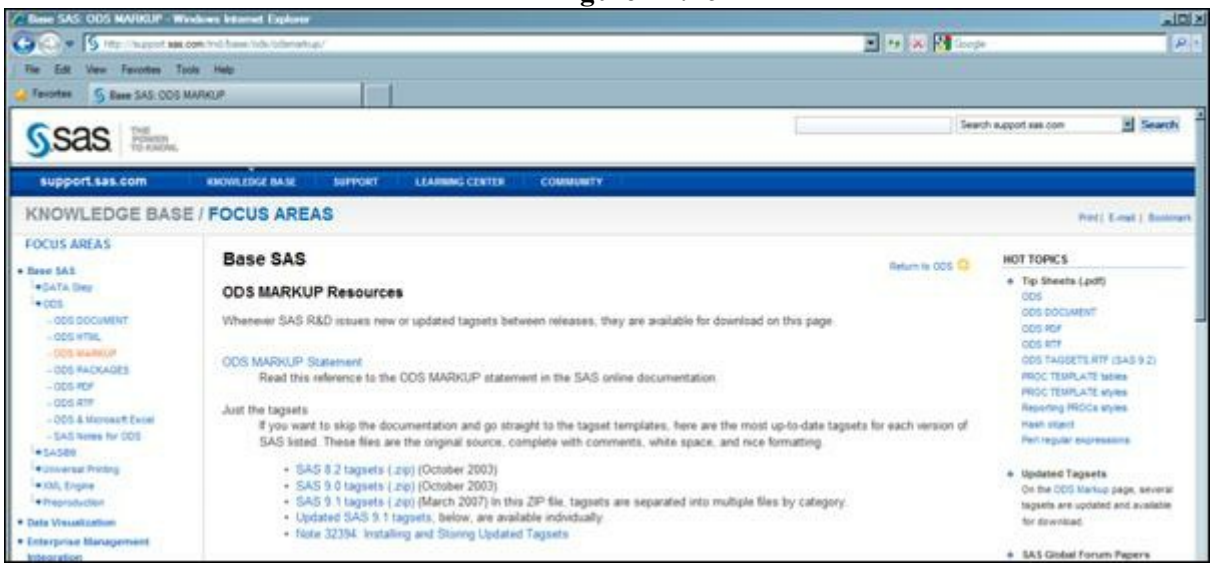


Figure 12.17

Just the tagsets

If you want to skip the documentation and go straight to the tagset templates, here are the most up-to-date tagsets for each version of SAS listed. These files are the original source, complete with comments, white space, and nice formatting.

- SAS 8.2 tagsets (.zip) (October 2003)
- SAS 9.0 tagsets (.zip) (October 2003)
- SAS 9.1 tagsets (.zip) (March 2007) In this ZIP file, tagsets are separated into multiple files by category.
- Updated SAS 9.1 tagsets, below, are available individually.
- Note 32394: Installing and Storing Updated Tagsets

Figure 12.18

Updated SAS 9.1 Tagsets to Download Individually

CSV, CSVALL, CSVBYLINE (April 2006)

This May 2005 update to the SAS 9.1.3 tagset has enhanced features and performance over the released version. New to this tagset is a fix for some problems caused by a specific form of Proc Tabulate. Another fix has to do with treating currency as numbers or character strings. For examples, see the SUGI 31 paper The Beginner's Guide to ODS MARKUP: Don't Panic! (.pdf). The following ODS Statement will provide complete details of all the new features:

```
ODS CSV file="test.csv" options(doc="help")
```

ExcelXP *New update! April 2008 (version 1.86, 04/15/08)*

See the quick reference to options and change log for the tagset.

This new update to the SAS 9.1.3 tagset has enhanced features and performance over the released version. This new version has too many new features to list! Some highlights include better column widths and spacing control, control over many print options, generation of a table of contents or an index of worksheets, and worksheet intervals that break on bygroups. Output from the ExcelXP tagset also works with the Calc spreadsheet program from OpenOffice.org. The following ODS Statement will provide complete details of all the new features:

```
ODS tagsets.excelxp file="test.xml" options(doc="help")
```

Previous versions of the ExcelXP tagset are available:

- June 2007 (version 1.72, 06/09/07)
- May 2007 (version 1.62, 05/11/07)
- June 2006 (version 1.37, 05/31/06).

More about Using the Doc="Help" Suboption

As explained earlier in the chapter, almost all of the tagset templates have some kind of internal self-documentation. In this way, the suboptions that are supported by each tagset template can be changed, documented, and released for download on a more frequent update cycle than new releases of SAS. If you submit the following ODS CSV invocation, the following Help Text would appear in the SAS log, as shown in Figure 12.19.

```
ODS CSV FILE='result1.csv' options(Doc='Help');
```

Figure 12.19

```
================================================================================
The CSV Tagset Help Text.

This Tagset/Destination creates output in comma separated value format.

Numbers, Currency and percentages are correctly detected and show as numeric values.
Dollar signs, commas and percentages are stripped from numeric values by default.

================================================================================

These are the options supported by this tagset.

Sample usage:

ods csv options(doc='Quick');

ods csv options(currency_as_number='yes' percentage_as_number='yes' delimiter=';');

Doc:  No default value.
    Help: Displays introductory text and options.
    Quick: Displays available options.

Delimiter:   Default Value ','
    Sets the delimiter for the values.  Comma is the default.  Semi-colon is
    a popular setting for european sites.

currency_as_number:   Default Value 'No'
    If 'Yes' currency values will not be quoted.
    The currency values are stripped of punctuation and currency symbols
    so they can be used as a number.

percentage_as_number:   Default Value 'No'
    If 'Yes' percentage values will not be quoted.
```

The percentages are stripped of punctuation and the percent sign

so they can be used as a number.

Currency_symbol: Default Value '$'

Used for detection of currency formats and for

removing those symbols so excel will like them.

Will be deprecated in a future release when it is

no longer needed.

Decimal_separator: Default Value '.'

The character used for the decimal point.

Will be deprecated in a future release when it is no longer needed.

Prepend_Equals: Default Value 'no'

Put an equal sign in front of quoted number values.

This only works in conjunction with quote_by_type.

quote_by_type: Default Value 'no'

Put values based on the type, not based on what the regex match for number.

Table_Headers: Default Value 'yes'

If no, skip the header section of all tables.

Thousands_separator: Default Value ','

The character used for indicating thousands in numeric values.

Used for removing those symbols from numerics so excel will like them.

Will be deprecated in a future release when it is no longer needed.

Quoted_columns: Default Value ''

A list of column numbers that indicate which values should be quoted

ie. Quoted_columns="123"

Bylines: Default Value: No

If yes bylines will be printed

Titles: Default Value: No

If yes titles and footnotes will be printed

Notes: Default Value: No

If yes Note, Warning, Error, and Fatal notes will be printed

Proc_Titles: Default Value: No

If yes titles generated by the procedures will be printed

Part 4
Output Style

Chapter 13 Style Template Definitions 345
Chapter 14 Understanding the Built-In Style Template Definitions 357
Chapter 15 Modifying Output Fonts 399
Chapter 16 Modifying Output Structure 419
Chapter 17 Modifying Output Colors and Graphics 439
Chapter 18 Doing More with Styles and ODS 467
Chapter 19 Special Cases: The REPORT, TABULATE, and PRINT Procedures 493
Chapter 20 Understanding ODS Graphics Output 543

Chapter 13
Style Template Definitions

ODS is a major step forward in giving SAS users the power to customize output. The chapters so far have explained how to send output to various destinations and how to work with output objects.

The next few chapters are all about appearance. ODS gives you the power to customize virtually every aspect of the appearance of your results. You can make a title bigger, make the detail data bolder, or change the entire color scheme.

To make these customizations easier, ODS comes with a number of built-in style template definitions. This chapter covers how to use these existing style templates. We will switch from one style template to another, and make minor changes in an existing style template. You've already seen the results of using the simple STYLE= option. Subsequent chapters go over style template definition modifications in more detail and show how you can build your own custom style template definitions.

Style templates had significant syntax changes between SAS 9.1.3 and SAS 9.2. As in the rest of the book, this chapter uses SAS 9.2 syntax. See the author pages for this book at http://support.sas.com/authors for SAS 9.1.3 versions of the programs.

This chapter uses the data set FASHIONS to illustrate the concepts. This data set is also available on the author pages for this book.

Viewing a List of Available Style Template Definitions

You might not have realized it, but each of the preceding examples that produced HTML, RTF, or PRINTER output used a style template definition. By default, ODS uses a standard style template for each output destination. These default styles use simple fonts and various shades of blue, gray, and black (HTML), or gray and black (RTF and PRINTER). When you issue an ODS statement like:

```
ODS HTML BODY='DefaultStyle.html';
```

you're really issuing the following:

```
ODS HTML BODY='DefaultStyle.html' STYLE=Default;
```

So if you want to switch to another style template, all you have to do is add a STYLE= option and specify the name of a different style template.

What are these other style templates? There are two ways to get a list of the style templates on your system. The first way is to use the **Tools** option on the menu bar in Display Manager. From the **Tools** option, select **Options** and then **Preferences**. From the window that pops up, select the **Results** tab. In the middle of the **Results** tab is a drop-down list labeled **Style**. This list shows you the several style template definitions available on your system, as shown in Figure 13.1.

A second way to get a list of style templates is to use the LIST STYLES statement in PROC TEMPLATE. The following code generates a list of available styles. Figure 13.2 shows partial output from this step.

```
    PROC TEMPLATE;
  LIST STYLES;
RUN:
```

The first item in the list is simply the directory location of the style definitions that are being listed. The items after the first are names of style templates, and they all have the type Style. Notice the style template called Default. This style template is used whenever you specify HTML output. The style template called Printer is used by default when you ask for PRINTER(PDF, PCL, and PostScript) output, and the style template called RTF is used by default when you ask for RTF output.

You can guess at the uses for some of the other style template definitions. The style template Minimal is just that: a spare template you can use if you don't want anything fancy. FancyPrinter is a more elaborate Printer style template. The Brown and Beige style templates use those colors. When you run the preceding PROC TEMPLATE step, your list might be different because you might be running a different release of SAS or your site might include custom style templates created by other users.

Figure 13.1

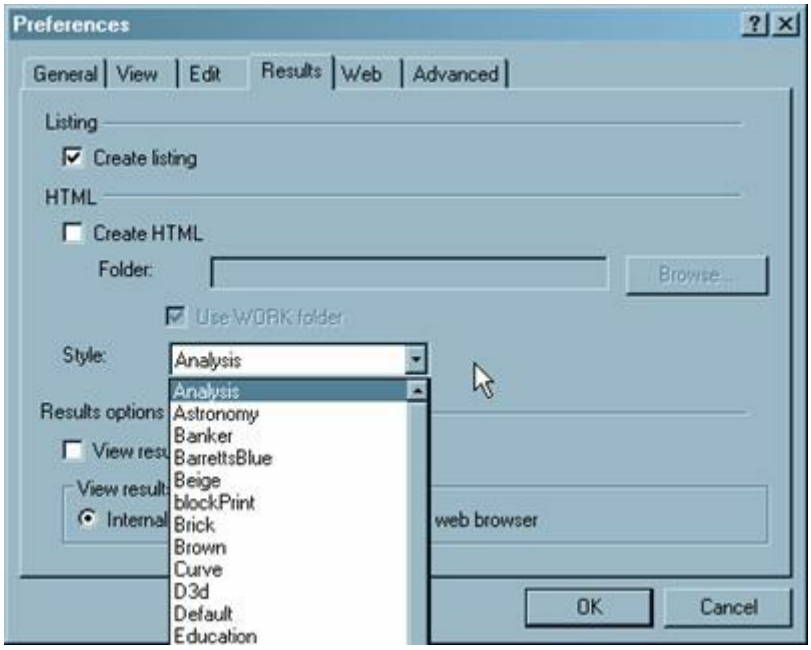


Figure 13.2

Listing of: SASHELP.TMPLMST
Path Filter is: Styles
Sort by: PATH/ASCENDING

Obs	Path	Type
1	Styles	Dir
2	Styles.Analysis	Style
3	Styles.Astronomy	Style
4	Styles.Banker	Style
5	Styles.BarrettsBlue	Style
6	Styles.Beige	Style
7	Styles.Brick	Style
8	Styles.Brown	Style
9	Styles.Curve	Style
10	Styles.D3d	Style
11	Styles.Default	Style
12	Styles.EGDefault	Style
13	Styles.Education	Style
14	Styles.Electronics	Style
15	Styles.Festival	Style
16	Styles.FestivalPrinter	Style
17	Styles.Gears	Style
18	Styles.Harvest	Style
19	Styles.HighContrast	Style
20	Styles.Journal	Style
21	Styles.Journal2	Style
22	Styles.Journal3	Style
23	Styles.Listing	Style
24	Styles.Magnify	Style
25	Styles.Meadow	Style
26	Styles.MeadowPrinter	Style
27	Styles.Minimal	Style
28	Styles.Money	Style
29	Styles.NoFontDefault	Style
30	Styles.Normal	Style
31	Styles.NormalPrinter	Style
32	Styles.Ocean	Style
33	Styles.Printer	Style
34	Styles.Rsvp	Style
35	Styles.Rtf	Style

Changing Style Template Definitions

Switching from one style template to another is quite simple. To switch to a new style template for your current SAS session, you can manually change the style template using the Preferences window as shown in the previous example. However, you have to remember to do this for each session. A better approach is to include the style template name in your program.

To switch from the default style template for a given output destination, all you do is add the STYLE= option to your ODS statement when you open the output destination. For example, if you wanted to create HTML output using the Analysis style template, you would add STYLE=Analysis to your code. The following example shows how to use the same procedure and apply different styles to change the appearance of the output. The code sends output to the PDF and HTML destinations.

The following ODS PDF statement applies the style Statdoc to the PROC MEANS report, and the results are shown in Figure 13.3.

```
      ODS PDF FILE='StatdocStyle.pdf' STYLE=Statdoc;
proc means data=fashions maxdec=0 sum;
  class Skirts;
  var Sales;
run;
ODS PDF CLOSE;
```

Figure 13.3 uses blue and various shades of gray, and if you have a color printer, this is a nice style template to use. You can also apply the style Statdoc to your HTML output by changing the ODS statement in the previous example. Figure 13.4 shows the HTML version of the output, and it looks similar to the PDF output.

```
ODS HTML BODY='StatdocStyle.htm' STYLE=Statdoc;
```

Now let's look at another style template by applying the style template Analysis.

```
ODS PDF FILE='AnalysisStyle.pdf' STYLE=Analysis;
```

Figure 13.5 shows the output using the Analysis style in SAS 9.2.

As a final example, let's look at how the Journal style looks when sending the PROC MEANS report to the PDF destination. Figure 13.6 presents the report. The Journal style was created for results that are to be published in professional journals, and it meets the style guidelines of many journals.

```
ODS PDF FILE='JournalStyle.pdf' STYLE=Journal;
```

Experiment with each of the style templates in each of the output formats, and learn which ones you like best for each type of output. Note that the styles used in this example are included with SAS®9. Earlier versions have fewer built-in styles.

Analysis Variable : sales Sales		
Skirt Length	N Obs	Sum
Above knees	194	11767742
Below knees	168	10068671
Maxi	75	4777235
Mini	73	3975498

Figure 13.3

Analysis Variable : sales Sales		
Skirt Length	N Obs	Sum
Above knees	194	11767742
Below knees	168	10068671
Maxi	75	4777235
Mini	73	3975498

Figure 13.4

Analysis Variable : sales Sales		
Skirt Length	N Obs	Sum
Above knees	194	11767742
Below knees	168	10068671
Maxi	75	4777235
Mini	73	3975498

Figure 13.5

Analysis Variable : sales Sales		
Skirt Length	*N Obs*	*Sum*
Above knees	194	11767742
Below knees	168	10068671
Maxi	75	4777235
Mini	73	3975498

Figure 13.6

Examining the Components of an ODS Style Template Definition

You don't have to know how style templates work in order to be able to use them. However, you do need to learn something about them if you want to be able to modify them. The SOURCE statement in PROC TEMPLATE can list the source code for a style template definition. The following step lists the source code for the Default style.

```
PROC TEMPLATE;
  SOURCE Styles.Default;
RUN;
```

This step produces a list in the SAS log that is hundreds of lines long. To keep things simple, we're just going to look at a small sample of the code. Figure 13.7 shows the lines of source code that affect the titles in your output. This includes the default SAS title ("The SAS System ...") and any titles you add with a TITLE statement.

The code begins with a DEFINE statement that names the style template it creates. The first section of code starts with **class fonts**, and it defines the fonts to use for titles and regular document text. This section is called a *style statement* (or *style class*) and it defines a *style element* which is in turn made up of a number of *style attributes*. If you look at the Styles.Default style template in earlier versions of SAS, you would see the keyword STYLE where the keyword CLASS appears in the SAS 9.2 version of the template. In the Default style template, both the titles and the document text are set to the <sans-serif> font. Helvetica and sans-serif are listed as backup fonts in case your system does not have the preferred font. The only difference between the title and document fonts is size (5 and 4 versus 3) and weight (bold italic versus regular).

The second style class, **color_list**, sets up the colors that the Default style will use in creating your output. The colors are assigned to names (fgA, bgA) that will be used later in the code. The colors themselves are indicated using RGB values given in hexadecimal. The values fgA and bgA represent a dark blue and a light gray respectively. The third class, **colors**, applies these colors to the foreground and background color settings for titles and document text. Only a few of the settings in the third style class are shown in Figure 13.7.

The fourth style class, **Container**, is used for overall settings. Here the fonts and colors for regular document text are assigned. This **Container** style class is also the main organizing element of the style template definition.

The fifth style class, **TitlesAndFooters**, is the specific part of the style template definition that defines what makes titles and footnotes different from other parts of the style template. The final style class, **SystemTitle**, has a special setting that applies only to titles. The font is changed from the TitleFont2 that was defined in **`TitlesAndFooters`** to the larger sized TitleFont.

Together, all of these style elements create a title with a particular font and colors. This was only an introduction, but it gives a basic feel for how style template definitions work. For more details on style template definitions, see the TEMPLATE procedure chapter in the

Output Delivery System User's Guide at http://support.sas.com/documentation.

Note that if you do not yet have SAS 9.2, you will see slightly different syntax in your template definitions. The older version of template syntax contains "from" inheritance references with the STYLE statement, such as **Style TitlesAndFooters from Container**.

Figure 13.7: Selected Sections of SAS 9.2 Default Style Template Definition Source Code

```
define style Styles.Default;
  class fonts
    "Fonts used in the default style" /
    'TitleFont2' = ("<sans-serif>, Helvetica, sans-serif",4,bold italic)
    'TitleFont' = ("<sans-serif>, Helvetica, sans-serif",5,bold italic)
    'headingFont' = ("<sans-serif>, Helvetica, sans-serif",4,bold)
    'docFont' = ("<sans-serif>, Helvetica, sans-serif",3)
     … additional fonts listed here;
  class color_list
    "Colors used in the default style" /
    'fgB1' = cx004488
    'fgA1' = cx000000
    'bgA1' = cxF0F0F0
    'fgA' = cx002288
    'bgA' = cxE0E0E0
     … additional colors listed here;
  class colors
    "Abstract colors used in the default style" /
    'titlefg' = color_list('fgA')
    'titlebg' = color_list('bgA')
    'systitlefg' = color_list('fgA')
    'systitlebg' = color_list('bgA')
    'docfg' = color_list('fgA')
    'docbg' = color_list('bgA');
  class Container /
    font = Fonts('DocFont')
    color = colors('docfg')
    backgroundcolor = colors('docbg');
  class TitlesAndFooters /
    font = Fonts('TitleFont2')
    backgroundcolor = colors('systitlebg')
    color = colors('systitlefg');
  class SystemTitle /
    font = Fonts('TitleFont');
```

The preceding code does not represent a complete style template definition. Major required portions of the code have been left out due to space considerations. Do not attempt to use this code with PROCTEMPLATE to create a style template definition.

Although the complete style template definition source code is long and intimidating, you might want to spend a little time reviewing it just to get a better feel for how a style template definition works. Don't worry about understanding all of the details, but take the time to explore the basics. This understanding will pay off as you start to modify style template definitions in Chapters 14 through 18.

Modifying a Style Template Definition: A Simple Example

Now that you've seen a simple example of how a style template definition works, let's look at a simple example of modifying one. We're going to create a new style template that produces output just like the Default style template does, except that our new one uses a different font for the titles.

The fonts in the previous example were defined in a class called "fonts." The full list of fonts is shown in Figure 13.7. To modify a specific font, we use PROC TEMPLATE. A DEFINE statement names our new style template, and a PARENT= statement indicates that the new style template will be based on the Default style template definition. The name of our new template is DefaultNewTitle.

```
    PROC TEMPLATE;
  DEFINE STYLE DefaultNewTitle;
  PARENT=styles.default;
```

The PARENT= statement makes it easy to create a new style template, because you don't have to rewrite the whole style template definition. All you have to do is indicate the parts you want to change from the style named in the PARENT= statement by inserting new versions of those parts. In this example, we replace the fonts class shown in Figure 13.7 with the following code. The only line that is changed in the definition is for **TitleFont**, where the code sets the font to Times New Roman. This line is highlighted in the code that follows. Finally, an END statement marks the end of the style class definition changes, and a RUN statement executes the procedure. After this new style template is stored, it can be used repeatedly.

```
class fonts /
  'TitleFont2' = ("<sans-serif>, Helvetica, sans-serif",4,bold italic)
  'TitleFont' = ("Times New Roman, Times, sans-serif",5,bold italic)
  'StrongFont' = ("<sans-serif>, Helvetica, sans-serif",4,bold)
  'EmphasisFont' = ("<sans-serif>, Helvetica, sans-serif",3,italic)
  'FixedEmphasisFont' = ("<monospace>, Courier, monospace",2,italic)
  'FixedStrongFont' = ("<monospace>, Courier, monospace",2,bold)
  'FixedHeadingFont' = ("<monospace>, Courier, monospace",2)
  'BatchFixedFont' = ("SAS Monospace, <monospace>, Courier, monospace",2)
  'FixedFont' = ("<monospace>, Courier",2)
  'headingEmphasisFont' = ("<sans-serif>, Helvetica, sans-serif",4,bold italic)
  'headingFont' = ("<sans-serif>, Helvetica, sans-serif",4,bold)
  'docFont' = ("<sans-serif>, Helvetica, sans-serif",3);
end;
```

To use the new style template, just reference it in the STYLE= option of your ODS statement. Figure 13.8 shows the Default style for HTML output from the same PROC MEANS step used in the previous example, with a TITLE statement added to the step. Figure 13.9 shows the output formatted with the DefaultNewTitle style.

The next three chapters show more on changing styles. Although our programs use SAS 9.2 syntax, the author pages for the book have SAS 9.1.3 versions of the style template code.

Figure 13.8

Sales Results

The MEANS Procedure

Analysis Variable : sales Sales		
Skirt Length	N Obs	Sum
Above knees	194	11767742
Below knees	168	10068671
Maxi	75	4777235
Mini	73	3975498

Figure 13.9

Sales Results

The MEANS Procedure

Analysis Variable : sales Sales		
Skirt Length	N Obs	Sum
Above knees	194	11767742
Below knees	168	10068671
Maxi	75	4777235
Mini	73	3975498

Finding Your Custom Style Template Definitions

Where do style template definitions go after you've created them? The previous example created the style DefaultNewTitle. We were then able to apply that style to HTML output using the STYLE=DefaultNewTitle option in the ODS HTML statement.

The output from PROC TEMPLATE with the LIST STYLES statement does not include your new DefaultNewTitle style, as shown in Figure 13.10. It's also not present when examining the list from the **Results** tab in the Preferences window.

The reason is that ODS saves your custom style template definitions in a SAS item store called TEMPLAT in the SASUSER library. The built-in style template definitions are stored in a SAS item store called SASHELP.TMPLMST. When you submit an ODS statement with a STYLE option, SAS looks in both places to find the style template you have requested. How does this work? ODS has a PATH statement that defines the locations for all style templates. By default, the PATH statement is set to the following:

```
ODS PATH SASUSER.TEMPLAT(UPDATE) SASHELP.TMPLMST(READ);
```

SAS looks in these two locations for style templates (as well as table templates and tagsets). In addition, this default setting identifies the TMPLMST item store as read-only (to protect the built-in style templates) and the TEMPLAT item store as one you can update. When you specify a style with the STYLE= option, SAS looks for it in both locations (by default).

The default output for the LIST STYLES statement in PROC TEMPLATE is also only the SASHELP.TMPLMST item store. To see your custom style template definitions, remove the STYLES option from the LIST statement and include a WHERE clause to select all styles. The following PROC TEMPLATE step lists the style templates in SASUSER.TEMPLAT.

```
    PROC TEMPLATE;
  LIST / WHERE=(TYPE="Style");
RUN;
```

Figure 13.11 shows the output from this step. One last way you can view your new style is to go to the template browser window by typing **odstemplates** in the command line. This brings up the tree view shown in Figure 13.12.

If you do not have Write access to the SASUSER library or if you start building a number of style template definitions, you might want to organize them into separate item stores. You can do this by using a PATH statement with a new item store name in the PROC TEMPLATE step when you create the new style template definitions. Then you can add an ODS PATH statement to your code when you need to use the new style templates. Another reason to create a new location for your custom style template definitions is that the SASUSER.TEMPLAT item store will get overwritten if you reinstall SAS. To keep your style template definitions safe, either save the code that created them or set up a different location for the item store.

Figure 13.10

8	Styles.Brown	Style
9	Styles.Curve	Style
10	Styles.D3d	Style
11	Styles.Default	Style
12	Styles.EGDefault	Style
13	Styles.Education	Style
14	Styles.Electronics	Style
15	Styles.Festival	Style
16	Styles.FestivalPrinter	Style
17	Styles.Gears	Style
18	Styles.Harvest	Style

Figure 13.11

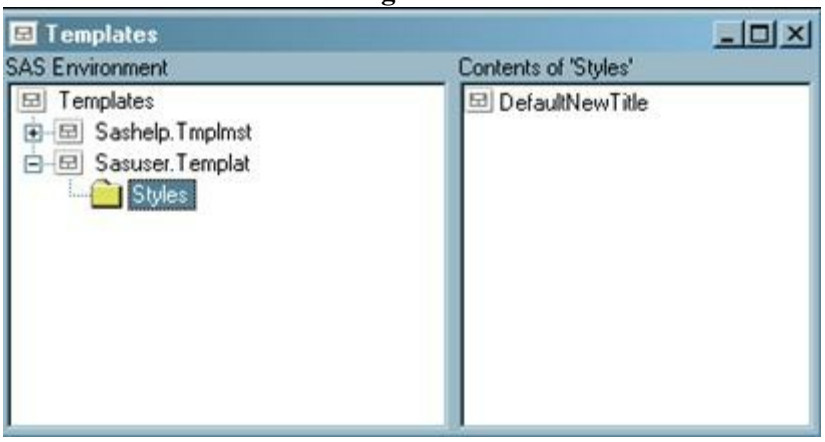
Listing of: SASUSER.TEMPLAT
Path Filter is: *
Sort by: PATH/ASCENDING

Obs	Path	Type
1	DefaultNewTitle	Style

Listing of: SASHELP.TMPLMST
Path Filter is: *
Sort by: PATH/ASCENDING

Obs	Path	Type
1	Base.Template.Style	Style
2	Styles.Analysis	Style
3	Styles.Astronomy	Style
4	Styles.Banker	Style
5	Styles.BarrettsBlue	Style

Figure 13.12



Chapter 14
Understanding the Built-In Style Template Definitions

A SAS installation includes a set of built-in ODS style templates that SAS saves in the SASHELP.TMPLMST item store. This item store is where you also find the table templates, graph templates, and tagset templates that SAS ships. Style templates work with ODS destinations that support style, and these include the HTML, RTF and PRINTER-family destinations.

A style template is a collection of style elements that are linked to pieces of tabular output such as headers and row headers or linked to general attributes such as colors and fonts. A style template can also have “abstract” elements such as Container or Document, which exist so that other style elements can inherit common attributes from them.

Style elements are a convenient way to name a group of style attributes for easy reference. If you are familiar with CSS style specifications, SAS style elements are roughly equivalent to CSS class selectors.

This chapter uses the data sets BILLINGS and HR to illustrate the concepts. These data sets are available on the author pages for this book at http://support.sas.com/authors.

Understanding the Default Style in SAS 9.1.3

The basic inheritance model for style templates changed between SAS 9.1.3 and SAS 9.2. This change allowed the developers to simplify how style templates could be changed. Even if you have SAS 9.2, this section might help you understand the background of style templates.

SAS has a single default style template, Styles.Default, that it uses differently depending on whether you send output to the HTML, RTF, or PDF destinations. This style template produces the familiar gray and blue ODS HTML output. SAS also uses it as the basis for RTF and Printer-family styles, and the style has specific modifications for these destinations.

As you saw in a previous chapter, you can investigate the style elements and attributes used for a particular piece of output with the ODS MARKUP diagnostic tagsets. Although there is more to a style template than just colors and fonts, using the STYLE_POPUP tagset to see that information is a good place to start. Recall this example from Chapter 12, the ODS MARKUP chapter:

```
ODS TAGSETS.STYLE_POPUP FILE='StyleDiag.html';
proc print data=billings(obs=5);
  title "Demonstrating Diagnostic Tagsets";
run;
ODS _ALL_ CLOSE;
```

Shown in Figure 14.1, hovering your mouse over the SAS title causes the background color of the title to change and a pop-up box to appear. The box contains the alignment and HTMLCLASS information for the element, which in this case is the SystemTitle element. If you double-click in the highlighted area, the browser displays a separate pop-up window that contains attributes for the SystemTitle element. Figure 14.2 shows this pop-up window.

To see the source code for Styles.Default and examine the SystemTitle element, submit the following PROC TEMPLATE step. The procedure writes the code in the SAS log.

```
PROC TEMPLATE;
  SOURCE styles.default;
RUN;
```

The SystemTitle element definition for Styles.Default is shown at the bottom of Figure 14.3, and it contains only an attribute for the font. However, the pop-up window in Figure 14.2 contains several attributes. The SystemTitle element inherits some of its style attributes from the TitlesandFooters style element. The pop-up window shows you all the style attributes used in the SAS title after inheritance has been resolved. By tracing all the ancestor references for SystemTitle as shown by the lines and arrows in Figure 14.3, you can discover all the values seen in the pop-up window.

The two color attributes are foreground and background. The four font attributes are FONT_FACE=, FONT_SIZE=, FONT_WEIGHT=, and FONT_STYLE=. The next examples show more about inheritance of the color and font attributes.

Figure 14.1

Figure 14.2

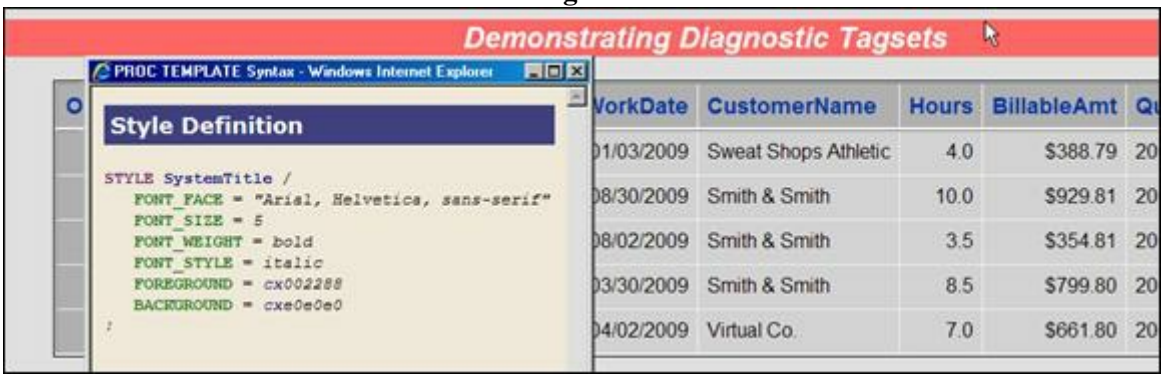


Figure 14.3: Selected Style Elements and Attributes from Styles.Default (SAS 9.1.3)

```
style fonts
   "Fonts used in the default style" /
   'TitleFont2' = ("Arial, Helvetica, sans-serif",4,Bold Italic)
   'TitleFont' = ("Arial, Helvetica, sans-serif",5,Bold Italic)
   'docFont' = ("Arial, Helvetica, sans-serif",3);

style color_list
   "Colors used in the default style" /
   'fgA' = cx002288
   'bgA' = cxE0E0E0;

style colors
   "Abstract colors used in the default style" /
   'systitlefg' = color_list('fgA')
   'systitlebg' = color_list('bgA')
   'docfg' = color_list('fgA')
   'docbg' = color_list('bgA');

style Container
   "Abstract. Controls all container oriented elements." /
   font = Fonts('DocFont')
   foreground = colors('docfg')
   background = colors('docbg');

style TitlesAndFooters from Container
   "Abstract. Controls system page title text and
              system page footer text." /
   font = Fonts('TitleFont2')
   background = colors('systitlebg')
   foreground = colors('systitlefg');

style SystemTitle from TitlesAndFooters
   "Controls system title text." /
   font = Fonts('TitleFont');
```

Understanding the Default Style in SAS 9.2

SAS 9.2 introduced a new style inheritance structure. The style template ancestor in SAS 9.2 for *all* style templates is Base.Template.Style, and as shown in Figure 14.4, the source code is documented with the text that it acts as the implicit parent. The Default style inherits definitions from Base.Template.Style.

The following PROC TEMPLATE step displays in the SAS log the source code for Base.Template.Style. Figure 14.4 shows the code from the first DEFINE statement to the section that contains the SystemTitle style element.

```
PROC TEMPLATE;
  SOURCE BASE.TEMPLATE.STYLE;
RUN;
```

The SystemTitle element in the Base.Template.Style definition is similar to the SAS 9.1.3 SystemTitle element in that it uses the same FROM syntax, and it is different in that it does not specify any style attributes. The definition for Base.Template.Style contains only the template inheritance model, which is essentially all of the FROMs.

In our upcoming examples, we feature SAS 9.2 syntax. The author pages for this book contains SAS 9.1.3 versions of the style template programs.

Figure 14.4: Base.Template.Style Partial

```
7    PROC TEMPLATE;
8      SOURCE BASE.TEMPLATE.STYLE;
define style Base.Template.Style;
   notes "Implicit parent for all style templates";
   style StartUpFunction
      "Controls the StartUp Function. TAGATTR is only element used.";
   style ShutDownFunction
      "Controls the Shut-Down Function. TAGATTR is only element used.";
   style Container
      "Controls all container oriented elements." /
      abstract =| on;
   style Document from Container
      "Controls the various document bodies.";
   style Body from Document
      "Controls the Body/Frame/Contents/Page file.";
   style Frame from Document
      "Controls the Body/Frame/Contents/Page file.";
   style Contents from Document
      "Controls the Body/Frame/Contents/Page file.";
   style Pages from Document
      "Controls the Body/Frame/Contents/Page file.";
   style Date from Container
      "Controls how date fields look." /
      abstract =| on;
   style BodyDate from Date
      "Controls the date field in the Body/Contents/Pages file.";
   style ContentsDate from Date
      "Controls the date field in the Body/Contents/Pages file.";
   style PagesDate from Date
      "Controls the date field in the Body/Contents/Pages file.";
   style IndexItem from Container
      "Controls list items and folders for Contents and Pages." /
      abstract =| on;
   style ContentFolder from IndexItem
      "Controls the generic folder definition in the Contents file.";
   style ByContentFolder from ContentFolder
      "Controls the byline folder in the Contents file.";
   style ContentItem from IndexItem
      "Controls the leafnode item in the Contents/Pages file.";
   style PagesItem from IndexItem
      "Controls the leafnode item in the Contents/Pages file.";
   style Index from Container
      "Controls Contents and Pages." /
      abstract =| on;
   style IndexProcName from Index
      "Controls the proc name in the list files." /
      abstract =| on;
   style ContentProcName from IndexProcName
      "Controls the proc name in the Contents file.";
   style ContentProcLabel from ContentProcName
      "Controls the proc label in the Contents file.";
   style PagesProcName from IndexProcName
      "Controls the proc name in the Pages file.";
   style PagesProcLabel from PagesProcName
      "Controls the proc label in the Pages file.";
   style IndexAction from IndexItem
      "Determines what happens on mouse-over events for folders and items.
   style FolderAction from IndexAction
      "Determines what happens on mouse-over events for folders.";
   style IndexTitle from Index
      "Controls the title of Contents and Pages files." /
      abstract =| on;
   style ContentTitle from IndexTitle
      "Controls the title of the Contents/Pages file.";
   style PagesTitle from IndexTitle
      "Controls the title of the Contents/Pages file.";
   style SysTitleAndFooterContainer from Container
      "Controls container for system page title and system page footer.";
   style TitleAndNoteContainer from Container
      "Controls container for procedure defined titles and notes.";
   style TitlesAndFooters from Container
      "Controls system page title text and system page footer text." /
      abstract =| on;
   style BylineContainer from Container
      "Controls container for the byline.";
   style SystemTitle from TitlesAndFooters
      "Controls system title text.";
```

Understanding the Default Style in SAS 9.2 (continued)

The SAS 9.2 code for Styles.Default still contains specific style attribute information, as shown in Figure 14.5. This output presents the same set of style elements that define the SystemTitle element, as was shown in Figure 14.3.

The most obvious difference between Figure 14.3 and Figure 14.5 is the presence of the CLASS statement in the SAS 9.2 version of Styles.Default. This is a new PROC TEMPLATE statement introduced in SAS 9.2. The statement's primary function works to your advantage when you create or modify a style template, because it creates a style element by looking for any like-named element in your template definition's parent template, if there is one. If not, it looks in Base.Template.Style.

Because Base.Template.Style is the implicit parent for any new template, you don't have to worry about the FROM syntax of the STYLE statement as you did in SAS 9.1.3. Additionally, this new template inheritance model makes the REPLACE statement in PROC TEMPLATE unnecessary.

In SAS 9.1.3, you could use the REPLACE statement to bypass the STYLE/FROM syntax when modifying a template. Inheritance was different if you used STYLE/FROM syntax or the REPLACE statement. Because this difference was confusing to users, developers added the CLASS statement in SAS 9.2 and simplified the inheritance model. To read more on development of the SAS 9.2 model, refer to SAS documentation on the SAS Web site.

SAS 9.2 syntax is featured in this chapter and subsequent chapters. However, the author pages for this book contain SAS 9.1.3 versions of any programs that use specific SAS 9.2 syntax.

Figure 14.5: Selected Style Elements and Attributes from Styles.Default in SAS 9.2

```
class fonts
    "Fonts used in the default style" /
    'TitleFont2' = ("<sans-serif>, Helvetica, sans-serif",4,bold italic)
    'TitleFont' = ("<sans-serif>, Helvetica, sans-serif",5,bold italic)
    'docFont' = ("<sans-serif>, Helvetica, sans-serif",3);

class color_list
    "Colors used in the default style" /
    'fgA' = cx002288
    'bgA' = cxE0E0E0;

class colors
    "Abstract colors used in the default style" /
    'systitlefg' = color_list('fgA')
    'systitlebg' = color_list('bgA')
    'docfg' = color_list('fgA')
    'docbg' = color_list('bgA');

class Container /
    font = Fonts('DocFont')
    color = colors('docfg')
    backgroundcolor = colors('docbg');

class TitlesAndFooters /
    font = Fonts('TitleFont2')
    backgroundcolor = colors('systitlebg')
    color = colors('systitlefg');

class SystemTitle /
    font = Fonts('TitleFont');
```

Understanding the Color Style Elements

This section describes the color style elements in Styles.Default and how styles inherit attributes from other styles. The color style attributes for SystemTitle come from the TitlesandFooters style element.

Figure 14.3 and Figure 14.5 show that the TitlesandFooters element contains the attribute definition for foreground and background colors and that the element does not explicitly name the colors. Instead, the CLASS statement in TitlesandFooters assigns colors defined by style element colors. The text values enclosed in parentheses, systitlefg and systitlebg, are defined in colors, and these attributes in turn are assigned values by style element color_list.

The pop-up window in Figure 14.2 shows that the value for background color is cxe0e0e0, and the value for foreground color is cx002288. These two values are hexadecimal numbers that represent the colors expressed as percentages of red, green and blue.

Figure 14.6 shows the color_list and colors classes from the Styles.Default code.

The style element color_list creates a list of color values. Each color value is assigned an alias. The color names that start with “fg” are generally assigned to foreground colors and those that start with “bg” are generally assigned to background colors. The color_list code associates color cxe0e0e0 with alias bgA and color cx002288 with alias fgA. The colors code associates fgA with systitlefg and fgB with systitlebg.

The style element colors defines longer names that reference locations in the output, and it links these names to the shorter names in the color_list element, which in turn specify the hexadecimal values of the colors.

This lookup process to trace the inheritance for every style element of interest to you is not necessary if you use the STYLE_POPUP tagset. Additionally, the diagnostic tagset ODSSTYLE can also “flatten” or resolve inheritance for you. The ODSSTYLE tagset specified in the following program saves the resolved code in the STYLESHEET= file.

```
ODS TAGSETS.ODSSTYLE FILE='ResolvedDefn.txt'
                STYLESHEET='ResolvedDefn.sas';
proc print data=billings(obs=5);
run;
ODS _ALL_ CLOSE;
```

Figure 14.7 shows the section for SystemTitle as copied from the STYLESHEET= file. (Even though the resolved inheritance code is put into the STYLESHEET= file, you do need a FILE= option with a dummy name in order for the code to execute correctly.) The ODSSTYLE output shows all the attributes, even if the font attributes are in a slightly different form. The next example discusses fonts and font usage within the default style definition.

Figure 14.6

```
class color_list
    "Colors used in the default style" /
    'fgB2' = cx0066AA
    'fgB1' = cx004488
    'fgA4' = cxAAFFAA
    'bgA4' = cx880000
    'bgA3' = cxD3D3D3
    'fgA2' = cx0033AA
    'bgA2' = cxB0B0B0
    'fgA1' = cx000000
    'bgA1' = cxF0F0F0
    'fgA' = cx002288
    'bgA' = cxE0E0E0;
class colors
    "Abstract colors used in the default style" /
    'headerfgemph' = color_list('fgA2')
    'headerbgemph' = color_list('bgA2')
    'headerfgstrong' = color_list('fgA2')
    'headerbgstrong' = color_list('bgA2')
    'headerfg' = color_list('fgA2')
    'headerbg' = color_list('bgA2')
    'datafgemph' = color_list('fgA1')
    'databgemph' = color_list('bgA3')
    'datafgstrong' = color_list('fgA1')
    'databgstrong' = color_list('bgA3')
    'datafg' = color_list('fgA1')
    'databg' = color_list('bgA3')
    'batchfg' = color_list('fgA1')
    'batchbg' = color_list('bgA3')
    'tableborder' = color_list('fgA1')
    'tablebg' = color_list('bgA1')
    'notefg' = color_list('fgA')
    'notebg' = color_list('bgA')
    'bylinefg' = color_list('fgA2')
    'bylinebg' = color_list('bgA2')
    'captionfg' = color_list('fgA1')
    'captionbg' = color_list('bgA')
    'proctitlefg' = color_list('fgA')
    'proctitlebg' = color_list('bgA')
    'titlefg' = color_list('fgA')
    'titlebg' = color_list('bgA')
    'systitlefg' = color_list('fgA')
    'systitlebg' = color_list('bgA')
    'Conentryfg' = color_list('fgA2')
    'Confolderfg' = color_list('fgA')
    'Contitlefg' = color_list('fgA')
    'link2' = color_list('fgB2')
    'link1' = color_list('fgB1')
    'contentfg' = color_list('fgA2')
    'contentbg' = color_list('bgA2')
    'docfg' = color_list('fgA')
    'docbg' = color_list('bgA');
```

Figure 14.7

```
style SystemTitle/
    Background = #E0E0E0
    Foreground = #002288
    ContentPosition = left
    Font = ("Arial, Helvetica, sans-serif", 5, bold italic)
    ContentScrollbar = auto
    BodyScrollbar = auto
;
```

Understanding the Font Style Elements

This section describes the font style attributes in Styles.Default. Two style elements control fonts: fonts and GraphFonts. Figure 14.8 shows the fonts and GraphFonts elements.

The Default style uses the fonts element for tabular reports. It uses the GraphFonts element for graphics driver output that supports style template definitions. This example focuses only on the fonts element for tabular output.

Recall from Figure 14.2 that the font attributes for the SystemTitle element are as follows:

FONT_FACE = *"Arial, Helvetica, sans-serif"*
FONT_SIZE = *5*
FONT_WEIGHT = *bold*
FONT_STYLE = *italic*

The TitleFont attribute in the fonts style shows the same set of attributes, but arranged differently.

("Arial, Helvetica, sans-serif",5,Bold Italic)

This specification is an alternate way to specify font attributes. The FONT_FACE= attribute is first and enclosed in quotation marks. The FONT_SIZE= attribute in this case is specified as the relative value of 5. Last are the FONT_WEIGHT= and FONT_STYLE= attributes, and these two values are not separated with a comma.

The FONT_FACE= value in this instance is not one font, but instead a list of fonts specified in the order in which ODS should use them. If the Arial font is not available when the ODS output file is opened, then ODS attempts to use Helvetica, and if Helvetica is not available, then ODS uses one of the system's sans-serif fonts. Finally, if no sans-serif font is available, the rendering software (browser or word processor) uses the software product's default font.

In Figure 14.8, the first font face for TitleFont is <sans-serif> or <monospace>, rather than a specific font. In order to support Unicode fonts, the registry associates a descriptive name with a particular font face. For example, the registry associates <monospace> with Courier New and <sans-serif> with Arial. This means that a user would only have to update the SAS registry instead of designing new templates if the user wanted to default to a monospace font other than Courier New.

Figure 14.8

```
class fonts
   "Fonts used in the default style" /
   'TitleFont2' = ("<sans-serif>, Helvetica, sans-serif",4,bold italic)
   'TitleFont' = ("<sans-serif>, Helvetica, sans-serif",5,bold italic)
   'StrongFont' = ("<sans-serif>, Helvetica, sans-serif",4,bold)
   'EmphasisFont' = ("<sans-serif>, Helvetica, sans-serif",3,italic)
   'FixedEmphasisFont' = ("<monospace>, Courier, monospace",2,italic)
   'FixedStrongFont' = ("<monospace>, Courier, monospace",2,bold)
   'FixedHeadingFont' = ("<monospace>, Courier, monospace",2)
   'BatchFixedFont' = ("SAS Monospace, <monospace>, Courier, monospace",2)
   'FixedFont' = ("<monospace>, Courier",2)
   'headingEmphasisFont' = ("<sans-serif>, Helvetica, sans-serif",4,bold italic)
   'headingFont' = ("<sans-serif>, Helvetica, sans-serif",4,bold)
   'docFont' = ("<sans-serif>, Helvetica, sans-serif",3);
class GraphFonts
   "Fonts used in graph styles" /
   'GraphDataFont' = ("<sans-serif>, <MTsans-serif>",7pt)
   'GraphUnicodeFont' = ("<MTsans-serif-unicode>",9pt)
   'GraphValueFont' = ("<sans-serif>, <MTsans-serif>",9pt)
   'GraphLabelFont' = ("<sans-serif>, <MTsans-serif>",10pt,bold)
   'GraphFootnoteFont' = ("<sans-serif>, <MTsans-serif>",10pt)
   'GraphTitleFont' = ("<sans-serif>, <MTsans-serif>",11pt,bold)
   'GraphAnnoFont' = ("<sans-serif>, <MTsans-serif>",10pt);
```

Understanding the HTML and Text Style Elements

The Styles.Default style template includes a few general housekeeping style elements that are described in this section. The HTML style element defines attributes that are used in the creation of ODS HTML output files. The Text style element defines attributes that are used elsewhere in an output file. Figure 14.9 shows both style elements.

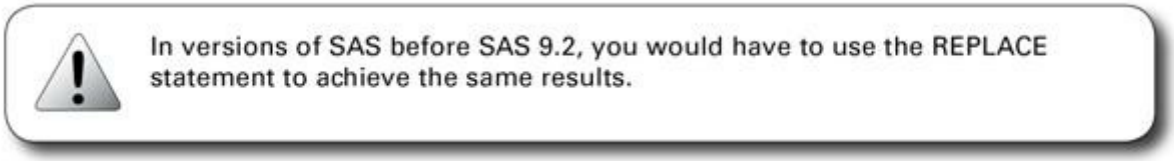
As an example of the Text element's usage, the values of style attributes PREFIX1= and SUFFIX1= in the Text element become part of element ContentProcName that appears in the HTML table of contents. (The specification for ContentProcName is not part of the HTML element.) The PREFIX1= and SUFFIX1= values are not affected by ODS NOPTITLE, which suppresses the procedure title in the output file itself. On the other hand, however, ODS PROCLABEL does override these values, because ODS PROCLABEL changes the high-level node in ODS-generated table of contents files.

In the HTML element, the PageBreakLine element inserts a horizontal rule (<HR>) where SAS intended to insert a page break in the output. Because HTML is a destination without pages or page breaks, the presence of this attribute might seem unnecessary. However, it serves as a way to tell where pages would be if the output were sent to a destination that could process page breaks.

The following code modifies the HTML element to increase the relative size of the page break line from 3 to 5 and to make the line red. Figure 14.10 shows how this HTML output would look when two PROC MEANS steps are run on the data set HR. The red line separates the two tables produced by PROC MEANS.

```
class html
  "Common HTML text used in the default style" /
  'expandAll' = "<span onclick=""if(msie4==1)expandAll()""">"
  'posthtml flyover line' = "</span><hr size=""3""">"
  'prehtml flyover line' = "<span><hr size=""3""">"
  'prehtml flyover bullet' = %nrstr("<span><b>&#183;</b>")
  'posthtml flyover' = "</span>"
  'prehtml flyover' = "<span>"
  'break' = "<br>"
  'Line' = "<hr size=""3""">"
  'PageBreakLine' = '<p style="page-break-after: always;">
                    <br></p><hr size="5" color="red">'
  'fake bullet' = %nrstr("<b>&#183;</b>");
```

Some expertise with HTML tags is required before you change these attributes. ODS does not use the HTML element attributes for the RTF or PRINTER destinations.



In versions of SAS before SAS 9.2, you would have to use the REPLACE statement to achieve the same results.

Figure 14.9: HTML and Text Style Elements

```
class html
   "Common HTML text used in the default style" /
   'expandAll' = "<span onclick=""expandCollapse()"">"
   'posthtml flyover line' = "</span><hr size=""3""">"
   'prehtml flyover line' = "<span><hr size=""3""">"
   'prehtml flyover bullet' = %nrstr("<span><b>&#183;</b>")
   'posthtml flyover' = "</span>"
   'prehtml flyover' = "<span>"
   'break' = "<br>"
   'Line' = "<hr size=""3""">"
   'PageBreakLine' = "<p style=""page-break-after: always;""><br></p><hr size=""3""">"
   'fake bullet' = %nrstr("<b>&#183;</b>");
class text
   "Common text." /
   'continued' = "(Continued)"
   'Fatal Banner' = "Fatal:"
   'Error Banner' = "Error:"
   'Warn Banner' = "Warning:"
   'Note Banner' = "Note:"
   'Pages Title' = "Table of Pages"
   'Content Title' = "Table of Contents"
   'suffix1' = " Procedure"
   'prefix1' = "The ";
```

Figure 14.10

USA	131	131	49110.69	35000.00	133500.00

The SAS System

The MEANS Procedure

Analysis Variable : AnnualSalary Annual Salary					
Position Category	**N Obs**	**N**	**Mean**	**Minimum**	**Maximum**
Contract	65	65	55592.31	35000.00	140000.00

Understanding the Abstract Style Elements

Some of the ODS style elements inherit attributes from abstract style elements. The only purpose of an abstract style element is to provide an element from which many other style elements can inherit their attributes. The HeadersAndFooters element in Styles.Default is an example of an abstract element. Figure 14.11 compares the definition of the HeaderAndFooters element from Styles.Default in SAS 9.2 versus SAS 9.1.3.

In the SAS 9.2 Styles.Default template, you see only a CLASS statement for the HeadersAndFooters element, the Header element, and the RowHeader element. In order to see the rest of the inheritance model, you have to look in the Base.Template.Style template, which contains the inheritance model for style templates in SAS 9.2. Figure 14.11 also shows the corresponding definitions for these elements in Base.Template.Style. However, the SAS 9.1.3 template shows the inheritance and the definition in one place.

Both the Header and the RowHeader elements inherit some of their characteristics from the HeadersAndFooters element. The fact that no style attributes are listed under Header and RowHeader means that they inherit all their attributes from the HeaderAndFooters element.

There are two different ways to write a program to change all the style elements that inherit from the HeadersAndFooters elements. Both ways are illustrated in the following code. One way is to use the STYLE statement with a FROM, and another way is to use a CLASS statement.

```
PROC TEMPLATE;
  DEFINE STYLE styles.chgabs;
  PARENT=styles.default;
  . . . insert STYLE or CLASS statement here . . .
  END;
RUN;
```

The advantage of using the following STYLE statement is that the code would produce the same results in SAS 9.2 and in SAS 9.1.3:

```
STYLE HeadersandFooters from HeadersandFooters/
   font = ("Courier New, SAS Monospace",4,Bold)
   background=yellow
   foreground=purple;
```

This step changes both the Header element and the RowHeader element because their parent element, HeadersAndFooters, was changed. Although this PROC TEMPLATE step would execute correctly in SAS 9.2, you might prefer to use the CLASS statement instead of the STYLE statement, as shown below.

```
CLASS HeadersandFooters /
   font = ("Courier New, SAS Monospace",4,Bold)
   background=yellow
   foreground=purple;
```

The style Chgabs produces Figure 14.12 when the style is applied to a PROC TABULATE report of the data set HR. The same output is created from either the STYLE statement or the CLASS statement.

It is impossible to document all of the abstract elements here. The ODS documentation has an excellent chart that shows all of the style elements and their progenitor elements. If you search in the SAS®9 documentation for the topic "Style Elements and Their Inheritances," you will find a comprehensive table of all the style elements used by ODS templates.

Figure 14.11: Styles.Default Abstract Element HeadersAndFooters

9.2 Styles.Default	9.1.3 Styles.Default

`CLASS HeadersAndFooters /`
`font = fonts('HeadingFont')`
`color = colors('headerfg')`
`backgroundcolor = colors('headerbg');`
`CLASS Header;`
`CLASS RowHeader;`
	`STYLE HeadersAndFooters from Cell`
	`"Abstract. Controls table headers and footers."/`
	`font = fonts('HeadingFont')`
	`color = colors('headerfg')`
	`backgroundcolor = colors('headerbg');`
	`STYLE Header from HeadersAndFooters`
	`"Controls the headers of a table.";`
	`STYLE RowHeader from Header`
	`"Controls row headers.";`

Base.Template.Style is the SAS 9.2 template that shows other elements that inherit from the HeadersAndFooters element.
`STYLE HeadersAndFooters from Cell`
`"Controls table headers and footers.";`
`STYLE Header from HeadersAndFooters`
`"Controls the headers of a table.";`
`STYLE RowHeader from Header`
`"Controls row headers.";`
	Base.Template.Style was not used for the inheritance model in SAS 9.1.3, so Styles.Default contains both the element definitions and the inheritance model.

Figure 14.12

Country				
Country	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Canada	173	43.58	173	43.58
Mexico	93	23.43	266	67.00
USA	131	33.00	397	100.00

In SAS 9.2, these are all equivalent statements:

```
STYLE HeadersandFooters from HeadersandFooters / . . .;
STYLE HeadersandFooters from _self_ / . . .;
STYLE HeadersandFooters / . . .;
```

Changing Specific Style Elements

The previous section described abstract style elements and how other elements inherit attributes from abstract style elements. You can also change a specific style element as well as the abstract element from which it inherits attributes. For example, the PROC TEMPLATE code that follows defines style DIFFHDRS, which is based on Styles.Default, and changes both the abstract style element Header and the style element RowHeader.

```
PROC TEMPLATE;
  DEFINE STYLE styles.diffhdrs;
    PARENT=styles.default;
    CLASS Header /
      font = ("Arial, Helvetica, sans-serif",5,Bold)
      background=black
      foreground=white
      just=center;
    CLASS RowHeader /
      font = ("Courier New, SAS Monospace",4,Bold)
      background=white
      foreground=black
      just=right;
  END;
RUN;
ODS PDF FILE="DiffHdrs.pdf" STYLE=STYLES.DIFFHDRS;
proc freq data=hr;
  tables department;
run;
ODS PDF CLOSE;
```

Figure 14.13 shows how each of the elements uses the different attributes.

In the preceding code, we changed the Header element and the RowHeader elements separately from each other because they specified different attributes. If the attributes are the same, however, you can list the two elements on the same CLASS or STYLE statement as follows. When specifying multiple elements, the FROM _SELF_ specification is needed.

```
CLASS Header, RowHeader from _self_ /
  font = ("Arial, Helvetica, sans-serif",5,Bold)
  background=black
  foreground=white
  just=center;
```

Figure 14.13

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Administration	70	17.63	70	17.63
Manufacturing	203	51.13	273	68.77
Marketing	72	18.14	345	86.90
R&D	52	13.10	397	100.00

In versions of SAS before SAS 9.2, you might have seen the REPLACE statement used instead of the STYLE statement in the PROC TEMPLATE code. For more information about changes between earlier versions of SAS and SAS 9.2, refer to the paper "The TEMPLATE Procedure Styles: Evolution and Revolution" by Kevin Smith, which can be found on the SAS Web site.

Startup and Shutdown Elements

If you have ever wondered how to make windows pop open on an HTML page, the trick is JavaScript or some other scripting language. Because Web developers frequently want to perform scripting functions when a page opens or closes, ODS includes the StartupFunction and ShutDownFunction style elements that you can add to a style template to pop open and close windows.

The TAGATTR= attribute specified with either of these functions passes your script code to your template. One useful function is the ALERT function, which you can use to pop open a window when the HTML page first loads. The following PROC TEMPLATE code defines a style so that a pop-up window with a message is displayed when the OVERTIME_ALERT HTML report is opened.

```
PROC TEMPLATE;
 DEFINE STYLE styles.alert;
   PARENT=styles.default;
   CLASS StartupFunction / tagattr='alert(
"To Get Paid For Overtime, Fill Out Your TimeSheets!!")';
 END;
RUN;
ODS HTML BODY='OvertimeAlert.html' STYLE=STYLES.ALERT;
title 'In-House Billing Overtime';
proc print data=Billings noobs label;
   where Hours>8 and JobType = 'In-house';
   label Hours='Overtime Hours';
   var EmployeeName WorkDate JobType Hours;
run;
ODS HTML CLOSE;
```

The StartupFunction style element places an empty JavaScript placeholder into your ODS HTML result file. The TAGATTR= style attribute specifies the JavaScript, which in this case is the ALERT function. Figure 14.14 shows the results of running this code. Note that you must not block pop-up windows in order to see the alert.

If you want to run a JavaScript program to add text to the body of your HTML document, you can use the PREHTML= and POSTHTML= attributes within the Body style element of your template to specify the JavaScript program. The PREHTML= attribute adds code to the beginning of a file, and the POSTHTML= attribute adds code to the end of the file. The following PROC TEMPLATE step defines a style that executes the JavaScript program GETDATE. This JavaScript program inserts date information at the end of the file. Figure 14.15 shows the results of using the Putdate style when running the same PROC PRINT step above.

```
PROC TEMPLATE;
  DEFINE STYLE styles.putdate;
    PARENT=styles.default;
    CLASS body / posthtml='<SCRIPT LANGUAGE=JAVASCRIPT
            TYPE="text/javascript" SRC="GETDATE.JS"></script>';
  END;
RUN;
```

Figure 14.14

In-House Billing Overtime

Employee Name	Date Work Completed	Job Type	Overtime Hours
Platt			11.0
Platt			8.5
Platt			9.0
Haworth			9.5
Haworth	07/15/2010	In-house	9.5

Windows Internet Explorer

To Get Paid For Overtime, Fill Out Your TimeSheets!!

OK

Figure 14.15

Platt	08/31/2009	In-house	11.0
Platt	05/05/2010	In-house	10.5
Knight	06/18/2010	In-house	10.5

From JavaScript Function:Thu Jan 29 15:52:06 2009

Check the SAS Technical Support Web site for additional references about StartupFunction usage. Some of the topics include the following:

- Resizing Window
- Loading Enterprise Guide Banner
- Set the ODS HTML Browser Always on Top
- Print ODS HTML File When Loaded Into Browser

Changing HTML Frames, Indexes, and Tables of Contents

This section describes how you can alter the HTML frame that ODS produces by default. It shows how to modify the style template elements that are specific to the generation of the frame, the index, and the table of contents (or table of pages). An alternative is to rearrange the output objects in an ODS DOCUMENT store, as discussed in Chapter 11, “Modifying and Replaying ODS Output.”

The list of style elements that can affect HTML frame navigation files is lengthy. This example modifies only a few. Table 14.1 lists the elements that this example modifies. For information about the other elements, consult the ODS documentation.

For this example, assume that we need to change the default placement, text, color, and font size of the contents area to that assigned in the following PROC TEMPLATE step. Additionally, we need to move the table of contents from the left side of the frame area to the bottom of the frame area. The Frame style element places the table of contents at the bottom.

```
PROC TEMPLATE;
  DEFINE STYLE styles.bottomtoc;
    PARENT=styles.default;
    CLASS Frame / contentposition=bottom  contentsize=25%;
    CLASS Contents / background=white;
    CLASS ContentTitle / font = ("Arial, Helvetica",16 pt,Bold)
                         foreground=black  background=white
              pretext = 'Select Your Report From the List Below';
    CLASS ContentFolder / background=white;
    CLASS ContentProcname / background=white  bullet=none;
    CLASS ContentItem / bullet=none  background=white;
  END;
RUN;
ODS HTML BODY= BODY='HRBody.html' CONTENTS='HRContents.html'
       FRAME='HRFrame.html' STYLE=styles.bottomtoc NEWFILE=proc;
proc freq data=hr;
  tables country;
run;
proc means data=hr n mean maxdec=2;
  class country;
  var annualsalary;
run;
proc means data=hr n mean min max maxdec=1;
  class department;
  var hoursweek;
run;
ODS HTML CLOSE;
```

Figure 14.16 shows part of the output from applying this style to a report containing output from the three PROC steps.

Table 14.1

Style Element	Description	Change
Frame	Generates the frame file attributes.	Move location of contents to bottom of frame area and increase space allocated to 25%.
Contents	Generates the contents file attributes.	BACKGROUND=WHITE
ContentTitle	Inserts the title in the contents file.	Change font, text string, background, and foreground colors.
ContentFolder	Controls the generic folder definition in the contents file.	BACKGROUND=WHITE
ContentProcname	Inserts the procedure name in the contents file.	BULLET=NONE BACKGROUND=WHITE
ContentItem	Inserts the lowest level of the hierarchy in a contents file.	BULLET=NONE BACKGROUND=WHITE

Figure 14.16

Consult with your Web site administrator before using ODS HTML frame navigation. He or she might prefer to get a list of files and build the navigation mechanism with a scripting language like JavaScript or PHP.

Changing HTML Body Style Attributes

This example describes how to modify the attributes in the Styles.Default HTML Body style. The Styles.Default template contains only three style attributes for the body of an ODS HTML document: LEFTMARGIN= (or MARGINLEFT=), RIGHTMARGIN= (or MARGINRIGHT=) and PAGEBREAKHTML=. Figure 14.17 shows the specifications for the Body style.

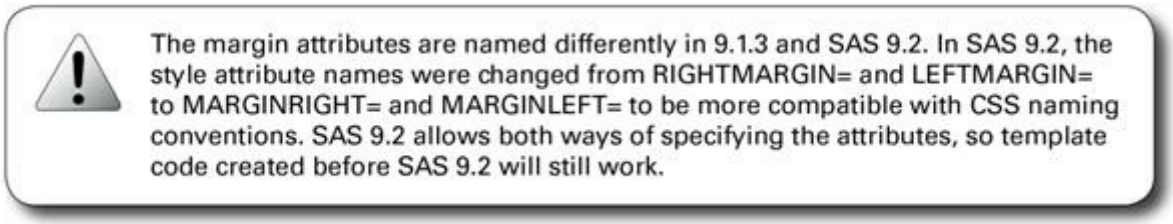
The following PROC TEMPLATE step changes the right margin from 8 pixels to 2 pixels and the left margin from 8 pixels to 0 pixels. It does not modify PAGEBREAKHTML=, so its setting remains the same as was defined in Styles.Default.

```
PROC TEMPLATE;
  DEFINE STYLE styles.chgbody;
  PARENT=styles.default;
    CLASS Body / marginright = 2
                 marginleft = 0;
  end;
run;
```

The SAS NOCENTER option must be in effect to see how setting the left margin to 0 affects the HTML output. The following program creates two HTML files with the same contents, but each formatted with a different style. ODS applies the new style Chgbody to the output in the first file and the Default style to the output in the second file.

```
OPTIONS NOCENTER;
ODS HTML(id=1) BODY='ChgBody.html' STYLE=styles.chgbody;
ODS HTML(id=2) BODY='DefBody.html' STYLE=styles.default;
ODS NOPTITLE;
proc means data=hr n mean min median max maxdec=2;
  class department;
  var annualsalary;
run;
ODS _ALL_ CLOSE;
```

Figure 14.18 presents the output ODS creates using the Chgbody template. The output is flush with the left-hand side of the browser screen. Figure 14.19 presents the output ODS creates using Styles.Default. This output is indented 8 pixels from the edge of the browser screen. If you had centered the output, the difference in margin would not have been discernable.



The margin attributes are named differently in 9.1.3 and SAS 9.2. In SAS 9.2, the style attribute names were changed from RIGHTMARGIN= and LEFTMARGIN= to MARGINRIGHT= and MARGINLEFT= to be more compatible with CSS naming conventions. SAS 9.2 allows both ways of specifying the attributes, so template code created before SAS 9.2 will still work.

Figure 14.17

```
class Body /
   marginright = 8
   marginleft = 8
   pagebreakhtml =
       html('PageBreakLine');
```

Figure 14.18

Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00
Manufacturing	203	203	51507.39	35000.00	37000.00	128000.00
Marketing	72	72	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

Figure 14.19

Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00
Manufacturing	203	203	51507.39	35000.00	37000.00	128000.00
Marketing	72	72	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

Changing Table Style Elements (Part 1 of 2)

Most SAS procedures produce tabular output. The main style element that controls this tabular output is the Table style element. The Table style element, however, inherits all of its attributes from the Output style element, which is an abstract element. Figure 14.20 shows the Output element definition.

Because many of these style attributes interact with each other, it is useful to change only one or two at a time, review the changes, and then repeat the process until you reach the desired results. Remember that you need to check the results in all destinations to which you intend to send output. The following PROC TEMPLATE step changes the BACKGROUNDCOLOR=, RULES=, BORDERSPACING=, BORDERCOLOR=, and BORDERWIDTH= attributes in the Output style element. Figure 14.21 shows annotated results of applying the Chgtable style to a PROC MEANS report.

```
PROC TEMPLATE;
  DEFINE STYLE styles.chgtable;
    PARENT=styles.default;
    CLASS Output / bordercollapse = separate
                   backgroundcolor = white
                   rules = ALL
                   frame = BOX
                   cellpadding = 7
                   borderspacing = 5
                   bordercolor = black
                   borderwidth = 5;
  END;
RUN;
```

Some naming conventions have changed in SAS 9.2. For example, the attributes CELLSPACING= and BORDERSPACING= both refer to the space between each cell.

Figure 14.20

```
class Output /
   bordercollapse = separate
   backgroundcolor = colors('tablebg')
   rules = GROUPS
   frame = BOX
   cellpadding = 7
   borderspacing = 1
   bordercolor = colors('tableborder')
   borderwidth = 1;
```

Figure 14.21

```
class Output / bordercollapse = separate
               backgroundcolor = white
               rules = ALL
               frame = BOX
               cellpadding = 7
               borderspacing = 5
               bordercolor = black
               borderwidth = 5;
```

The border color of black with a width of 5 frames the exterior of the table.

RULES=ALL draws interior lines between all the rows and columns in the table using black as the border color. Attribute BORDERSPACING=5 causes the background color of white to appear around each cell.

Analysis Va						al Salary
Department	N Obs	N	Mean	Minimum	Median	Maximum
A		0	51014.29	35000.00	35500.00	143500.00
M		3	51507.39	35000.00	37000.00	128000.00
M		2	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

Changing Table Style Elements (Part 2 of 2)

Consult the ODS documentation for a comprehensive list of all the possible values for the RULES= and FRAME= attributes. In combination, they provide you many different possibilities for drawing interior and exterior table lines. For example, just one change to the Output style element in Chgtable can make the background color of the table disappear, as shown in Figure 14.22:

```
borderspacing=0
```

Figure 14.23 shows you eight of the different looks that you can achieve with changes to only two attributes: RULES= and FRAME=. The setting for BORDERSPACING= is 0 for these outputs.

Valid values for the RULES= attribute are ALL, COLS, GROUPS, NONE, and ROWS. Valid values for the FRAME= attribute are ABOVE, BELOW, BOX, HSIDES, LHS, RHS, VOID, and VSIDES. You can generate forty different appearances for tables by combining the five possible values of rules with the eight possible values of frame.

Generally, when you change the RULES= attribute, you should set BORDERSPACING=0 or CELLSPACING=0 so that the table background color does not interfere with the RULES= setting. The BORDERSPACING=0 setting prevents the table background color of white from appearing in the outputs in Figure 14.23.

Here is the PROC TEMPLATE step to create a template for the first output in Figure 14.23:

```
PROC TEMPLATE;
  DEFINE STYLE styles.rulesframe;
    PARENT=styles.default;
    CLASS Output / borderspacing=0
                   rules = rows
                   frame = void;
  END;
RUN;
```

The same PROC FREQ step of data set HR produced the eight tables in Figure 14.23. All outputs were sent to the HTML destination.

Figure 14.22

Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00

Figure 14.23

RULES=ROWS and FRAME=VOID

Department		
Department	Frequency	Percent
Administration	70	17.63
Manufacturing	203	51.13
Marketing	72	18.14
R&D	52	13.10

RULES=COLS and FRAME=VOID

Department		
Department	Frequency	Percent
Administration	70	17.63
Manufacturing	203	51.13
Marketing	72	18.14
R&D	52	13.10

RULES=NONE and FRAME=VOID

The FREQ Procedure

Department		
Department	Frequency	Percent
Administration	70	17.63
Marketing	72	18.14
R&D	52	13.10
Manufacturing	203	51.13

RULES=GROUPS and FRAME=VOID

Department		
Department	Frequency	Percent
Administration	70	17.63
Manufacturing	203	51.13
Marketing	72	18.14
R&D	52	13.10

RULES=NONE and FRAME=HSIDES

Department		
Department	Frequency	Percent
Administration	70	17.63
Manufacturing	203	51.13
Marketing	72	18.14
R&D	52	13.10

RULES=NONE and FRAME=VSIDES

Department		
Department	Frequency	Percent
Administration	70	17.63
Manufacturing	203	51.13
Marketing	72	18.14
R&D	52	13.10

RULES=NONE and FRAME=BOX

Department		
Department	Frequency	Percent
Administration	70	17.63
Manufacturing	203	51.13
Marketing	72	18.14
R&D	52	13.10

RULES=NONE and FRAME=ABOVE

Department		
Department	Frequency	Percent
Administration	70	17.63
Manufacturing	203	51.13
Marketing	72	18.14
R&D	52	13.10

Changing Header-Related and Data Cell-Related Style Elements (Part 1 of 2)

The previous examples described how to modify the borders and rules of a table. This example and the next one describe the header-related and data cell-related style elements that control the rest of your table. Most SAS procedure output has header cells that span the top of the table. Figure 14.24 maps style elements to the header and data components in two tables.

Both the HeaderAndFooters and Data elements inherit their attributes from Cell, an abstract style element. From there, their paths diverge, with many styles in turn inheriting attributes from them and modifying attributes for the specific element.

For example, the RowHeader element inherits attributes from the Headers element. The Headers element inherits attributes from HeadersAndFooters. The lineage starts with HeadersAndFooters inheriting attributes from Cell. The RowHeader element in Styles.Default is the same as the Header element. However, since it is a separate definition, you have the ability to change the RowHeader element to be different from the Header element.

The following PROC TEMPLATE step creates a style that inherits its attributes from Styles.Default and modifies the Table, Header, and Data elements to make a table with decorative borders. You can modify the borders in your table with four attributes: BORDERRIGHTSTYLE=, BORDERLEFTSTYLE=, BORDERTOPSTYLE=, and BORDERBOTTOMSTYLE=. Or, if you want to change all four borders to the same style, you can just use BORDERSTYLE=. Figure 14.25 shows, however, that not all applications interpret the border settings the same way.

```
PROC TEMPLATE;
  DEFINE STYLE styles.decorative;
    PARENT=styles.default;
    CLASS Table / rules=all frame=box
      borderspacing=5 borderwidth=5 bordercolor=black
      bordertopstyle=dotted  borderbottomstyle=dotted
      borderrightstyle=dotted  borderleftstyle=dotted;
    CLASS Header / background=gray foreground=white
                   font_face='Courier New'
      bordertopstyle=dashed borderbottomstyle=dashed
      borderrightstyle=dashed borderleftstyle=dashed;
    CLASS Data / background=white foreground=black
      bordertopstyle=dotted borderbottomstyle=dotted
      borderrightstyle=dotted borderleftstyle=dotted;
  END;
RUN;
```

The BORDERRIGHTSTYLE=, BORDERLEFTSTYLE=, BORDERTOPSTYLE=, and BORDERBOTTOMSTYLE= attributes are new for SAS 9.2.

Figure 14.24

Header

Data

Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00
Manufacturing	203	203	51507.39	35000.00	37000.00	128000.00
Marketing	72	72	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

Header

RowHeader

Data

Department		
Department	Frequency	Percent
Administration	70	17.63
Manufacturing	203	51.13
Marketing	72	18.14
R&D	52	13.10

Figure 14.25

HTML

RTF

PDF

Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00
Manufacturing	203	203	51507.39	35000.00	37000.00	128000.00
Marketing	72	72	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

Changing Header-Related and Data Cell-Related Style Elements (Part 2 of 2)

The previous example and the PROC MEANS tabular report in Figure 14.25 demonstrate that the Header element and the Data element together control all of the cells in the report. This example focuses on the specifics of the Data element.

Like the Header element, the Data element inherits many of its characteristics from the Cell element, which is an abstract element. The Cell element in turn inherits style attributes from the Container element. Figure 14.26 shows the definitions for the Container and Data elements. The code shows that the Data element overrides the Container element settings for background color and foreground color and that only the setting for FONT= style attribute in Data remains the same as that defined in Container.

To add CSS style properties compatibility, SAS allows either COLOR= or FOREGROUND= to assign foreground color, and it allows either BACKGROUNDCOLOR= or BACKGROUND= to assign the background color.

The following PROC TEMPLATE step defines a style that's less decorative than the previous example's template. It demonstrates modifying the Data element as well as the Table, Header, and RowHeader elements. The example shows that you can change RowHeader and Header elements independently from each other.

```
PROC TEMPLATE;
  DEFINE STYLE styles.manyparts;
    PARENT=styles.default;
    CLASS Table / rules=all
                  frame=box
                  borderspacing=0
                  borderwidth=2
                  cellpadding=1
                  bordercolor=yellow;
    CLASS Header / background=gray
                   foreground=white
                   font_face='Courier New';
    CLASS RowHeader / background=black
                      foreground=white;
    CLASS Data / background=white
                 foreground=black;
  END;
RUN;
```

Figure 14.27 shows how ODS reduces the spacing between the cells when you change CELLPADDING= from the Styles.Default value of 7 pixels to 1 pixel.

To see the impact of the different Header and RowHeader style elements, we need to apply this style template to a procedure like FREQ that uses the RowHeader element, as shown in Figure 14.28.

Figure 14.26

```
class Container /
   font = Fonts('DocFont')
   color = colors('docfg')
   backgroundcolor = colors('docbg');
class Data /
   color = colors('datafg')
   backgroundcolor = colors('databg');
```

Figure 14.27

Default CELLPADDING=7

Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00
Manufacturing	203	203	51507.39	35000.00	37000.00	128000.00
Marketing	72	72	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

CELLPADDING=1

Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00
Manufacturing	203	203	51507.39	35000.00	37000.00	128000.00
Marketing	72	72	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

Figure 14.28

Department		
Department	Frequency	Percent
Administration	70	17.63
Manufacturing	203	51.13
Marketing	72	18.14
R&D	52	13.10

Header

RowHeader

Data

Adding Images to Style Elements

If you want to add images to a style template, you need to consider the destination of your output file and the viewer that will open the destination file. For example, ODS places a pointer to an image in the HTML <IMG> tag when it creates an HTML file. When a browser opens the HTML file, the browser dynamically finds and inserts the image. Because images are dynamically inserted in an HTML file by the browser when it opens the HTML file, you must maintain access to the images for anyone who opens the file. On the other hand, ODS embeds an image in the file sent to the PDF or RTF destination. When the PDF or RTF file is opened, the images are already incorporated into the document. Your application must have access to the images when ODS creates the PDF or RTF file, but you do not need to have access to the images when you open the rendered file. Table 14.2 lists style attributes that can reference images.

Table 14.2: Selected Style Elements That Reference Images

Style Attribute	Function	Destination
PREIMAGE= POSTIMAGE=	Inserts an image before or after text in a cell or before or after a table.	HTML, RTF, PDF, PCL, PS
BACKGROUNDIMAGE=	Inserts a background image.	HTML, PDF, PCL, PS
WATERMARK=	Specifies that the background= image should be treated as a watermark and appear in a fixed location on the HTML page.	HTML

The value supplied to the first three style attributes is a file reference for the image file. The value for the watermark attribute, however, is either ON or OFF. When WATERMARK=ON, ODS uses the image specified for the BACKGROUNDIMAGE= attribute as a watermark. Although you can specify images at the cell level, you are more likely to use them at the document or table level, as shown in the two PROC TEMPLATE steps that follow.

Use Images

```
PROC TEMPLATE;
  DEFINE STYLE styles.addimage;
  PARENT=styles.default;
  CLASS Body / preimage= 'LH.gif';
  CLASS Table /
            postimage='draft.gif';
  END;
RUN;
```

Use Watermark

```
PROC TEMPLATE;
  DEFINE STYLE styles.watermark;
  PARENT=styles.default;
  CLASS Body /
      backgroundimage = 'LH.gif'
      watermark=on;
  END;
RUN;
```

Figure 14.29 shows the results of using the two different images when applying the style on the left to a PROC MEANS report. Figure 14.30 shows the results of adding a watermark when applying the style on the right. So that the Web browser finds the images when it loads

the HTML file, you need to either store the images in the same directory as the HTML page or specify the file reference attribute as a fully qualified URL to reference the image, such as http://www.wombat.com/imagedir/LH.gif.

Figure 14.29

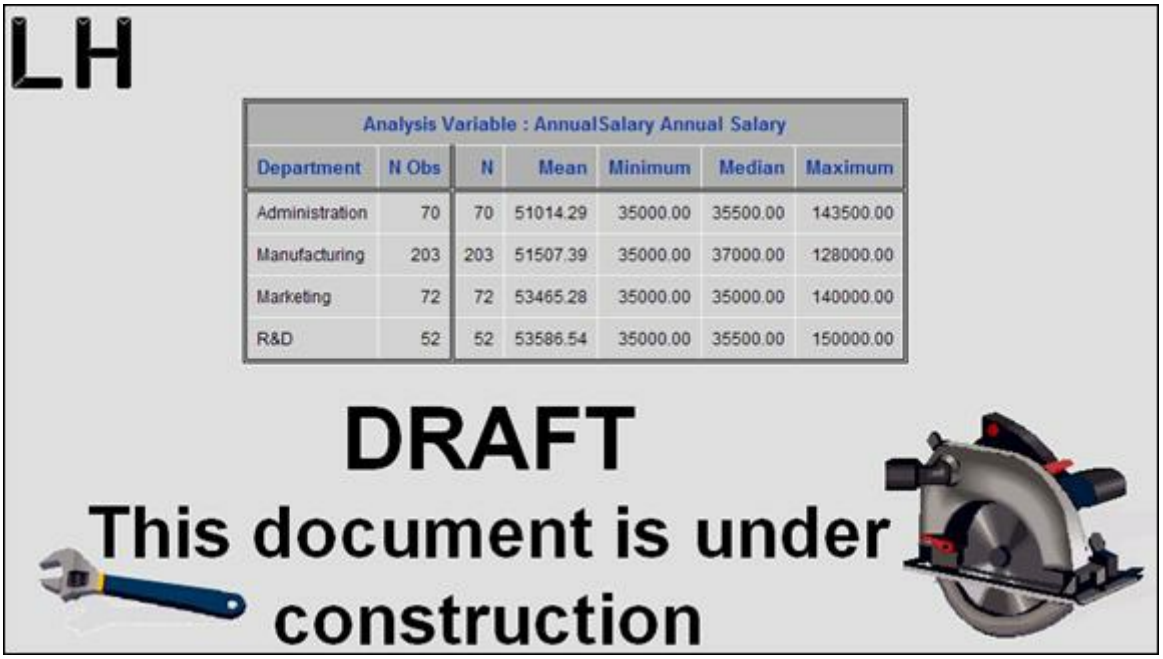
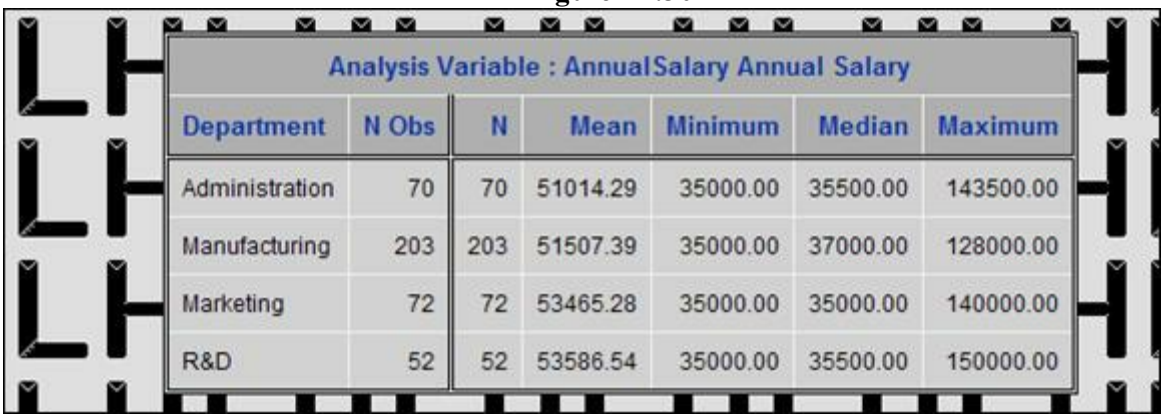


Figure 14.30



For more information about specifying images for HTML, RTF or PDF files, consult the SAS Technical Support Web site.

Modifying Titles and Footnotes Using Style Elements (Part 1 of 2)

The Styles.Default template has style elements that allow you to format each of the titles and footnotes separately. Figure 14.31 shows the inheritance path as defined in Base.Template.Style. Notice how each title or footnote element inherits from the one above it. For example, SystemTitle5 inherits from SystemTitle4, and so on, until you get to SystemTitle, which inherits its attributes from the TitlesandFooters element.

The following PROC TEMPLATE step illustrates how you can change each of the ten titles separately in a style template. Each footnote in this template inherits its attributes from the
same-numbered SAS title.

```
PROC TEMPLATE;
  DEFINE STYLE styles.fmttitle;
    PARENT=styles.default;
    CLASS SystemTitle / font_size=24pt;
    CLASS SystemTitle2 / font_size=22pt;
    CLASS SystemTitle3 / font_size=20pt;
    CLASS SystemTitle4 / font_size=18pt;
    CLASS SystemTitle5 / font_size=16pt;
    CLASS SystemTitle6 / font_size=14pt;
    CLASS SystemTitle7 / font_size=12pt;
    CLASS SystemTitle8 / font_size=10pt;
    CLASS SystemTitle9 / font_size=8pt;
    CLASS SystemTitle10 / font_size=6pt;
    CLASS SystemFooter from SystemTitle /;
    CLASS SystemFooter2 from SystemTitle2 /;
    CLASS SystemFooter3 from SystemTitle3/;
    CLASS SystemFooter4 from SystemTitle4 /;
    CLASS SystemFooter5 from SystemTitle5 /;
    CLASS SystemFooter6 from SystemTitle6 /;
    CLASS SystemFooter7 from SystemTitle7 /;
    CLASS SystemFooter8 from SystemTitle8 /;
    CLASS SystemFooter9 from SystemTitle9 /;
    CLASS SystemFooter10 from SystemTitle10 /;
  END;
RUN;
```

Figure 14.32 shows the output from a PROC MEANS step that uses the Fmttitle style and includes ten TITLE statements and ten FOOTNOTE statements.

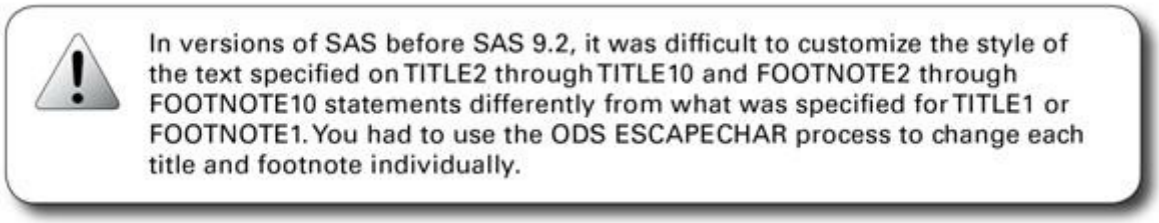
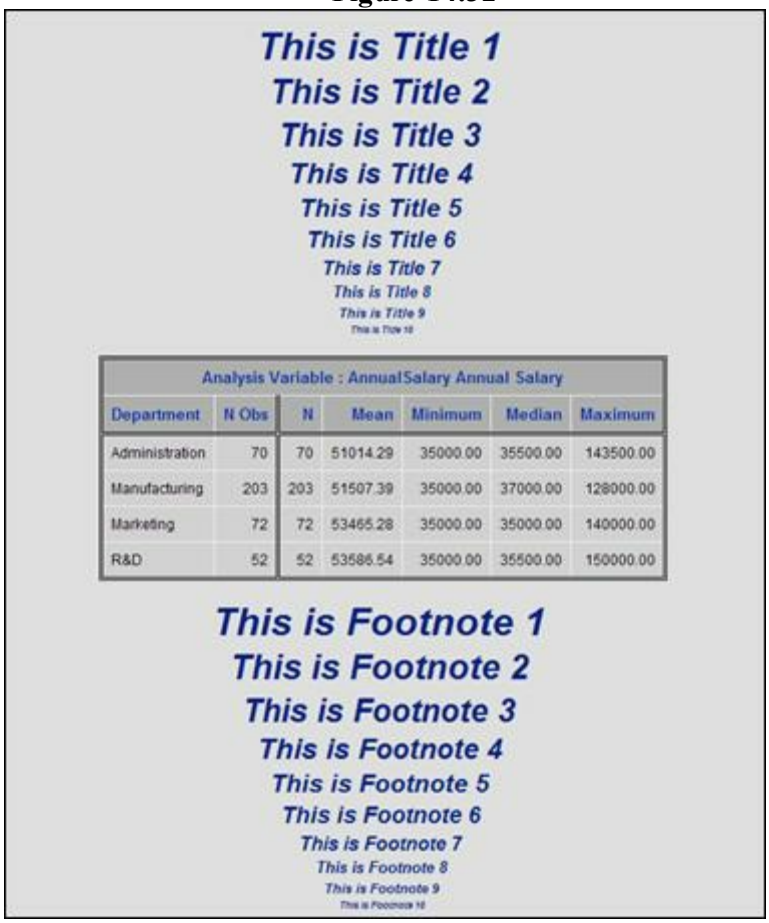

In versions of SAS before SAS 9.2, it was difficult to customize the style of the text specified on TITLE2 through TITLE10 and FOOTNOTE2 through FOOTNOTE10 statements differently from what was specified for TITLE1 or FOOTNOTE1. You had to use the ODS ESCAPECHAR process to change each title and footnote individually.

Figure 14.31`

```
style SystemTitle from TitlesAndFooters
style SystemTitle2 from SystemTitle
style SystemTitle3 from SystemTitle2
style SystemTitle4 from SystemTitle3
style SystemTitle5 from SystemTitle4
style SystemTitle6 from SystemTitle5
style SystemTitle7 from SystemTitle6
style SystemTitle8 from SystemTitle7
style SystemTitle9 from SystemTitle8
style SystemTitle10 from SystemTitle9
style SystemFooter from TitlesAndFooters
style SystemFooter2 from SystemFooter
style SystemFooter3 from SystemFooter2
style SystemFooter4 from SystemFooter3
style SystemFooter5 from SystemFooter4
style SystemFooter6 from SystemFooter5
style SystemFooter7 from SystemFooter6
style SystemFooter8 from SystemFooter7
style SystemFooter9 from SystemFooter8
style SystemFooter10 from SystemFooter9
```

Figure 14.32



Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00
Manufacturing	203	203	51507.39	35000.00	37000.00	128000.00
Marketing	72	72	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

Modifying Titles and Footnotes Using Style Elements (Part 2 of 2)

Consider a scenario similar to the previous example. Here, PROC TEMPLATE creates a style where it modifies only the ten SystemTitle styles and does not change the SystemFooter styles. The CLASS statements are the same as in the previous PROC TEMPLATE step.

```
PROC TEMPLATE;
  DEFINE STYLE styles.nofoot;
    PARENT=styles.default;
    CLASS SystemTitle / font_size=24pt;
    CLASS SystemTitle2 / font_size=22pt;
    CLASS SystemTitle3 / font_size=20pt;
    CLASS SystemTitle4 / font_size=18pt;
    CLASS SystemTitle5 / font_size=16pt;
    CLASS SystemTitle6 / font_size=14pt;
    CLASS SystemTitle7 / font_size=12pt;
    CLASS SystemTitle8 / font_size=10pt;
    CLASS SystemTitle9 / font_size=8pt;
    CLASS SystemTitle10 / font_size=6pt;
  END;
RUN;
```

Figure 14.33 shows that the title text is identically formatted as the title text in Figure14.32, whereas the footnote text is not. The SystemFooter elements retain their original inheritance from the TitlesAndFooters element. Therefore, Figure 14.33 shows that all the footnote text has the same font size.

The following modifications to the SystemTitle and SystemTitle2 elements demonstrate how you can insert an image and left-justify the image in the first title. It includes code to reset the two attributes PREIMAGE= and JUST=, because SystemTitle2 inherits all attributes set in SystemTitle1 except for those it explicitly changes. The default center for JUST= is CENTER. The _UNDEF_ keyword clears any specification for the PREIMAGE= attribute. The program does not include any text on the TITLE1 statement, so only the image appears in the first title location.

```
CLASS SystemTitle / preimage='LH.gif'
                    just=left font_size=24pt;
CLASS SystemTitle2 / preimage=_undef_
                     just=center font_size=22pt;
```

Figure 14.34 shows the results of these two changes to the Nofoot template. All other elements in the template remain the same. The FOOTNOTE statement text appears the same as in Figure 14.33.

Figure 14.33

This is Title 1
This is Title 2
This is Title 3
This is Title 4
This is Title 5
This is Title 6
This is Title 7
This is Title 8
This is Title 9
This is Title 10

Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00
Manufacturing	203	203	51507.39	35000.00	37000.00	128000.00
Marketing	72	72	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

This is Footnote 1
This is Footnote 2
This is Footnote 3
This is Footnote 4
This is Footnote 5
This is Footnote 6
This is Footnote 7
This is Footnote 8
This is Footnote 9
This is Footnote 10

Figure 14.34

LH

This is Title 2
This is Title 3
This is Title 4
This is Title 5
This is Title 6
This is Title 7
This is Title 8
This is Title 9
This is Title 10

Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00
Manufacturing	203	203	51507.39	35000.00	37000.00	128000.00
Marketing	72	72	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

This is Footnote 1
This is Footnote 2
This is Footnote 3
This is Footnote 4
This is Footnote 5
This is Footnote 6
This is Footnote 7
This is Footnote 8
This is Footnote 9
This is Footnote 10

Understanding the Differences in the ODS Printer Style

The Printer family of styles includes PS (PostScript), Printer, PCL (Printer Control Language) and PDF (Portable Document Format). The previous examples created mostly HTML output. The most obvious differences in the output created by the default style for the HTML destination and for the PDF destination are the colors and fonts. Figure 14.35 contrasts PDF and HTML output.

When ODS sends output to a Printer family destination using the default style, it sets attributes that make the output easy to read on the printed page. The Printer default style has a black, gray, and white color scheme, and ODS writes the text in serif fonts. Output sent to the HTML destination using the default style, on the other hand, creates output that is easy to read on a computer screen. The HTML default style has a gray and blue color scheme, and ODS writes the text in sans-serif fonts.

The default style for the Printer family is Styles.Printer. Its parent template is Styles.Default. The Styles.Printer style modifies only a few style elements that it inherits from Styles.Default, which are listed in Figure 14.36.

Of that subset of style elements, the ones of most interest are Table, color_list, and Body. Here are the definitions for these three elements:.

```
STYLE Table from Output / bordercollapse = separate
                    rules = ALL  cellpadding = 4pt
                    borderspacing = 0.25pt
                    borderwidth = 0.75pt;
STYLE color_list  / 'link' = blue    'bgH' = grayBB
                'fg' = black     'bg' = _undef_;
STYLE Body from Document / marginbottom = _undef_
                     margintop = _undef_
                     marginright = _undef_
                     marginleft = _undef_
                     pagebreakhtml = html('PageBreakLine');
```

The values for CELLPADDING=, BORDERSPACING=, and BORDERWIDTH= in the Table element are specified in point size, because that form is more appropriate to printed fonts than the relative font numbers used by HTML. The color_list element has fewer attributes than the Styles.Default version of color_list. Its colors are gray and black with blue for hyperlinks. The background color is undefined because there is really no point in having a background color for output to be printed. The margins in the Body element are undefined because the Printer family of destinations can use system or printer options for margins.

The remaining style elements in Styles.Printer are defined specifically for creation of Printer family output. Because of this, if you are designing a template for the PDF destination, it would be better to specify Styles.Printer as the parent of your new template rather than Styles.Default.

Figure 14.35

PDF Output

Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00
Manufacturing	203	203	51507.39	35000.00	37000.00	128000.00
Marketing	72	72	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

HTML Output

Analysis Variable : AnnualSalary Annual Salary						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	70	70	51014.29	35000.00	35500.00	143500.00
Manufacturing	203	203	51507.39	35000.00	37000.00	128000.00
Marketing	72	72	53465.28	35000.00	35000.00	140000.00
R&D	52	52	53586.54	35000.00	35500.00	150000.00

Figure 14.36

```
define style Styles.Printer;
   parent = styles.default;
   style fonts
   style GraphFonts
   style Table from Output
   style Graph from Graph
   style color_list
   style Body from Document
   style colors
   style GraphColors
   style ContentTitle from IndexTitle
   style PrintedContentsLabel
   style ContentItem from IndexItem
   style ContentFolder from IndexItem
   style ContentProcName from IndexProcName
   style ContentProcLabel from ContentProcName
   style Date from Container
   style GraphBox from GraphBox
   style GraphHistogram from GraphHistogram
   style GraphEllipse from GraphEllipse
   style GraphBand from GraphBand
   style GraphContour from GraphContour
```

Understanding Differences in the RTF Style

The RTF destination output is intended to be opened and rendered by word processing programs that support the RTF specification. The RTF destination has more in common with the PDF destination in terms of fonts and color schemes than with the HTML destination. It should not be a surprise, therefore, that the parent for the default style used for the RTF destination is Styles.Printer, whose parent in turn is Styles.Default. The default style for the RTF destination is Styles.RTF.

Because the parent of Styles.RTF is Styles.Printer, there are not many elements that change in Styles.RTF. Figure 14.37 lists the elements that do change, and these changes are minor. For example, the fonts element in Styles.RTF specifies <serif> or <monospace>, which are set in the SAS registry as Times New Roman or Courier New. On the other hand, Styles.Printer specifies <MTSerif>, which is set in the registry as Thorndate AMT. Figure 14.38 shows a view of the registry settings.

Another minor modification of Styles.Printer in creating Styles.RTF is a change in the Table style element. The Styles.RTF template specifies CELLPADDING=3pt, which is slightly different from the CELLPADDING=4pt setting in Styles.Printer.

```
STYLE table from table / cellpadding = 3pt;
```

The way inheritance works is that CELLPADDING=3pt applies to tables sent to the RTF destination. All other Table style element attributes in Styles.RTF either come from those listed in the parent template (Styles.Printer), or for those not listed in Styles.Printer, from the grandparent template (Styles.Default).

A style element that appears in Styles.RTF and is not present in Styles.Printer is the change for the titleAndNoteContainer element as shown below:

```
STYLE titleAndNoteContainer from titleAndNoteContainer/
  width = _undef_;
```

Remember that in RTF output, ODS inserts the SAS title into the RTF output file in the header section of the word processing document. When you open the RTF file with a word processor, you would not want the width of the title in the header to be restricted by some size specification made by ODS. You want the width of the title in the header section to be able to change as the document margins and orientation change.

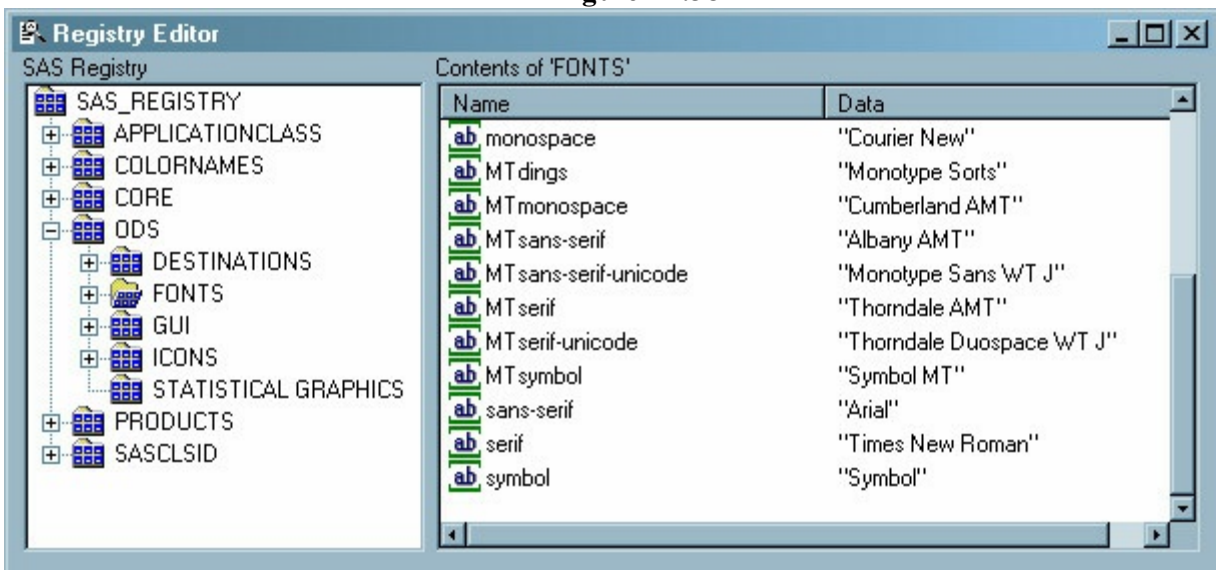
PDF and Printer files do not have the same concept of a header that word processing documents do. Therefore, the change in the titleAndNoteContainer is needed only in Styles.RTF. This specification in Styles.RTF overrides the attribute setting defined in the titleAndNoteContainer element in Styles.Default.

Style changes made in Styles.RTF are intrinsic to the RTF destination just as the changes needed in Styles.Printer are intrinsic for Printer family output. If you are designing a template for the RTF destination, it is best to specify Styles.RTF as the parent of your new template.

Figure 14.37

```
define style Styles.Rtf;
   parent = styles.printer;
   style Body from Document
   style fonts
   style GraphFonts
   style titleAndNoteContainer from titleAndNoteContainer
   style cell from container
   style table from table
   style batch from batch
   style Byline from Byline
   style color_list
   style GraphBox from GraphBox
   style GraphHistogram from GraphHistogram
   style GraphEllipse from GraphEllipse
   style GraphBand from GraphBand
```

Figure 14.38



Chapter 15
Modifying Output Fonts

This is the second chapter that shows how to modify the built-in style definitions. Rather than attempt to cover everything, these chapters focus on simple ways to modify the default style template. The examples demonstrate techniques that you can apply right away without having to take the time to thoroughly understand all there is to know about style definitions.

Each of the next three chapters deals with one aspect of style definitions, and this chapter describes fonts. This chapter includes a series of examples that show how to change the fonts in various parts of your results: titles, table headings, table cells, and footnotes. Aspects of fonts that are covered include how to change typefaces, font sizes, and font weights.

In our upcoming examples, we feature SAS 9.2 syntax. However, the author pages for this book at http://support.sas.com/authors contains SAS 9.1.3 versions of the style template programs.

This chapter uses the data set POSITIONS to illustrate the concepts. This data set is also available on the author pages for this book.

Identifying the Font Definitions for Each Part of Your Output

In the last chapter, we specified new font definitions for a title and some of the headers. We did this by modifying the default settings in style elements that affected the title and headers, such as HeadersAndFooters and RowHeader. The Styles.Default template also contains the fonts style element, which specifies the fonts for all parts of your output. If you modify the default settings in this element, the changes trickle down to all parts of your output. The settings for the fonts style element in Styles.Default are as follows:

```
'TitleFont2' 'TitleFont2' = ("<sans-serif>, Helvetica, sans-serif",4,bold italic)
'TitleFont' = ("<sans-serif>, Helvetica, sans-serif",5,bold italic)
'StrongFont' = ("<sans-serif>, Helvetica, sans-serif",4,bold)
'EmphasisFont' = ("<sans-serif>, Helvetica, sans-serif",3,italic)
'FixedEmphasisFont' = ("<monospace>, Courier, monospace",2,italic)
'FixedStrongFont' = ("<monospace>, Courier, monospace",2,bold)
'FiedHeadingFont' = ("<monospace>, Courier, monospace",2)
'BatchFixedFont' = ("SAS Monospace, <monospace>, Courier, monospace",2)
'FixedFont' = ("<monospace>, Courier",2)
'headingEmphasisFont' = ("<sans-serif>, Helvetica, sans-serif",4,bold italic)
'headingFont' = ("<sans-serif>, Helvetica, sans-serif",4,bold)
'docFont' = ("<sans-serif>, Helvetica, sans-serif",3);;
```

Although it's obvious what part of your output some of the attributes in the fonts element modify, for others it's not so obvious. For example, TitleFont sounds as though it should control the titles. However, it is not as clear what an attribute such as FixedFont controls. The way to find out which parts of the output each definition controls is to read the rest of the default style definition. For example, if you search for the definition EmphasisFont in Styles.Default, you find it used in the following seven style elements:

IndexTitle	ExtendedPage	DataEmphasis
HeaderEmphasis	RowHeaderEmphasis	FooterEmphasis
RowFooterEmphasis		

To save you the trouble of figuring out which part of the output is affected by each of the font definitions, Table 15.1 shows how Styles.Default uses the fonts. Since you will usually be working with the default style for your output destination, or with style definitions inherited from the default style, you can use Table 15.1 as a reference.

If you are working with a different style definition, such as Styles.Printer or Styles.RTF, the font definitions are slightly different. Continue to use Table 15.1 as a guideline. If you run into problems, review the source code for the style definition to identify the difference that is causing the problem.

If you are trying to modify a custom style definition created by another SAS developer, hopefully it will be based on the default style definition, and the information in Table 15.1

will still apply. If the custom style definition has substantial modifications, you might find that you need to review the source code in detail to learn how this custom template applies the font definitions.

Table 15.1 Output Component and Styles.Default and Fonts Element Attribute for Component

Output Part Controlled	Attribute in Fonts Element
Titles generated with TITLE statement	TitleFont
Titles for table of contents and table of pages	EmphasisFont
Titles for procedures ("The _____ Procedure")	TitleFont2
Titles for procedures that use fixed-width fonts	FixedStrongFont
Page numbers	StrongFont
Footnotes generated with FOOTNOTE statement	TitleFont
Table headings and footers	headingFont, FixedFont
Emphasized table headings and footers	EmphasisFont, FixedEmphasisFont
Strong (more emphasized) table headings and footers	StrongFont, FixedStrongFont
Table column and row headings and footers	headingFont, FixedFont
Emphasized table column and row headings and footers	EmphasisFont, FixedEmphasisFont
Strong (more emphasized) table column and row headings and footers	StrongFont, FixedStrongFont
BY group headings	headingFont
Data in table cells	docFont
Emphasized data in table cells	EmphasisFont, FixedEmphasisFont
Strong (more emphasized) data in table cells	StrongFont, FixedStrongFont
Batch mode output for procedures that use fixed-width font	BatchFixedFont
NOTE, WARNING, ERROR, and FATAL messages for procedures that use fixed-width font	FixedFont
Message when page overflows	EmphasisFont

Many of the output sections in Table 15.1 include two font definitions. The first definition listed is used for most of the procedures that you will encounter. However, there are a few procedures such as CHART and PLOT that still require fixed-width fonts in order to correctly format their results. For those procedures, a second set of fonts is provided. If you are modifying the font definitions, always replace a fixed-width font with another fixed-width font.

Modifying the Default Font Sizes for HTML Output

The default style definition for ODS HTML output, which is Styles.Default, uses some fairly large fonts for titles and table headings. This is great if you want a bold look and your titles and labels are short. However, if you have a lot of output to present, you might want to create a style definition that formats output in less space.

The previous example listed the contents of the fonts style element in Styles.Default. The items in parentheses completely define the font settings for the attribute. The first item in the list is the font typeface. The second item is a number that assigns relative font size. These numbers range from 2 for the smaller items in the output to 5 for the larger items.

Changing font sizes is easy. You start by running a PROC TEMPLATE step with a SOURCE statement to get the font list from the style definition that you want to modify. Then you write your PROC TEMPLATE step to modify selected default settings and create a new template. Your PROC TEMPLATE step needs a DEFINE statement to name the new template, a PARENT statement to specify the style template on which your new template is based, and a CLASS statement to define the new font settings.

The following PROC TEMPLATE step defines a style that makes the output text smaller by adjusting each of the default font sizes downward by 1. When we use the template DefaultSmaller, we should end up with output consistent with Styles.Default and proportionally smaller.

```
    PROC TEMPLATE;
  DEFINE STYLE DefaultSmaller;
    PARENT=Styles.Default;
    CLASS fonts /
      'TitleFont2'=("Arial, Helvetica, Helv",3,Bold Italic)
      'TitleFont'=("Arial, Helvetica, Helv",4,Bold Italic)
      'StrongFont'=("Arial, Helvetica, Helv",3,Bold)
      'EmphasisFont'=("Arial, Helvetica, Helv",2,Italic)
      'FixedEmphasisFont'=("Courier",1,Italic)
      'FixedStrongFont'=("Courier",1,Bold)
      'FixedHeadingFont'=("Courier",1)
      'BatchFixedFont'=("SAS Monospace, Courier",1)
      'FixedFont'=("Courier",1)
      'headingEmphasisFont'=("Arial,Helvetica,Helv",3,Bold Italic)
      'headingFont'=("Arial, Helvetica, Helv",3,Bold)
      'docFont'=("Arial, Helvetica, Helv",2);
  END;
RUN;
```

Compare Figure 15.1 to Figure 15.2. The PROC FREQ report on the data set HR in Figure 15.1 was created using Styles.Default, whereas the report in Figure 15.2 was created using the new template DefaultSmaller. Notice how the look of the two outputs is the same and that the font size has been scaled down proportionally in Figure 15.2.

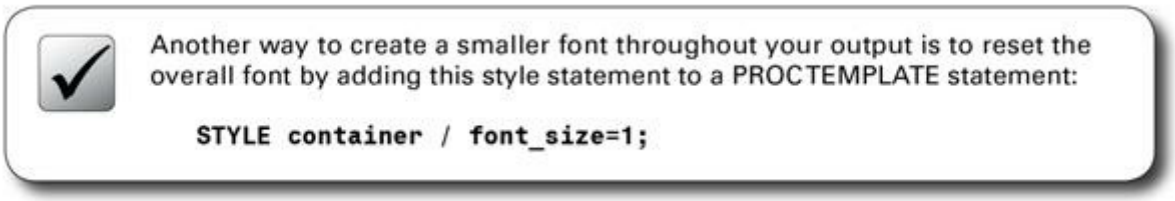


Figure 15.1

This wordy title does not fit on one line with STYLES.DEFAULT

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Administration	157	20.21	157	20.21
Sales	338	43.50	495	63.71
Technical	282	36.29	777	100.00

Figure 15.2

This wordy title does not fit on one line with STYLES.DEFAULT

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Administration	157	20.21	157	20.21
Sales	338	43.50	495	63.71
Technical	282	36.29	777	100.00

The font sizes used in Styles.Default range from 2 to 5. You can actually set them to values ranging from 1 to 7. These numbers represent relative font size codes used by HTML. ODS just passes these codes through to your output, where they are interpreted by a browser.

If you are using SAS 9.1.3, you need a REPLACE statement to supply the new font settings. If you are using SAS 9.2, you can use either a CLASS statement or a STYLE statement to define the new font settings.

Modifying the Default Font Sizes for Printer Output

When you want to change the font sizes for Printer output, you need to look at the default style definition for Printer output, which is Styles.Printer. Recall that Styles.Printer is based on Styles.Default.

By running PROC TEMPLATE with a SOURCE statement for Styles.Printer, you can find out how Styles.Printer specifies the attributes in its fonts element. Figure 15.3 shows the fonts element in Styles.Printer. There are two items to note in the fonts element. First, Styles.Printer is based on Styles.Default, as shown by the PARENT= statement. Second, Styles.Printer has its own list of font definitions, which are different from the font definitions in Styles.Default.

If we want output with a reduced font size, we need to create a new style template and specify new font sizes as we did in the previous example. Note that the font sizes in Styles.Printer are in points. This is different from the relative font sizes that Styles.Default specifies, which was shown when creating HTML output in the previous example.

To create a new style definition that uses smaller fonts, we copy the old font definitions, assign new point sizes that are adjusted downward, and name the new style template. The HTML example reduced font sizes by the relative value of 1. This example reduces the font sizes by 2 points for all settings except for BATCHFIXEDFONT= to achieve a similar proportional effect. (The setting for BATCHFIXEDFONT= was reduced by one-half point.)

The following PROC TEMPLATE step creates the template PrinterSmaller, which reduces the font sizes as described above.

```
    PROC TEMPLATE;
  DEFINE STYLE PrinterSmaller;
    PARENT=Styles.Printer;
  CLASS fonts / 'TitleFont2' = ("Times",10pt,Bold Italic)
    'TitleFont' = ("Times",11pt,Bold Italic)
    'StrongFont' = ("Times",8pt,Bold)
    'EmphasisFont' = ("Times",8pt,Italic)
    'FixedEmphasisFont' = ("Courier New, Courier",7pt,Italic)
    'FixedStrongFont' = ("Courier New, Courier",7pt,Bold)
    'FixedHeadingFont' = ("Courier New, Courier",7pt,Bold)
    'BatchFixedFont' = ("SAS Monospace,Courier New,Courier",6pt)
    'FixedFont' = ("Courier New, Courier",7pt)
    'headingEmphasisFont' = ("Times",9pt,Bold Italic)
    'headingFont' = ("Times",9pt,Bold)
    'docFont' = ("Times",8pt);
  END;
RUN;
```

Compare Figure 15.4 to Figure 15.5. The PROC FREQ report on the data set HR in Figure 15.4 was created using Styles.Printer, whereas the report in Figure 15.5 was created using the new template PrinterSmaller. Notice how the look of the two outputs is the same and

that the font size has been scaled down in Figure 15.5.

Figure 15.3

```
define style Styles.Printer;
  parent = styles.default;
  style fonts /
     'TitleFont2' = ("<MTserif>, Times Roman",12pt,bold italic)
     'TitleFont' = ("<MTserif>, Times Roman",13pt,bold italic)
     'StrongFont' = ("<MTserif>, Times Roman",10pt,bold)
     'EmphasisFont' = ("<MTserif>, Times Roman",10pt,italic)
     'FixedEmphasisFont' = ("<MTmonospace>, Courier",9pt,italic)
     'FixedStrongFont' = ("<MTmonospace>, Courier",9pt,bold)
     'FixedHeadingFont' = ("<MTmonospace>, Courier",9pt,bold)
     'BatchFixedFont' = ("SAS Monospace, <MTmonospace>, Courier",6.7pt)
     'FixedFont' = ("<MTmonospace>, Courier",9pt)
     'headingEmphasisFont' = ("<MTserif>, Times Roman",11pt,bold italic)
     'headingFont' = ("<MTserif>, Times Roman",11pt,bold)
     'docFont' = ("<MTserif>, Times Roman",10pt);
```

Figure 15.4

Department				
Department	**Frequency**	**Percent**	**Cumulative Frequency**	**Cumulative Percent**
Administration	157	20.21	157	20.21
Sales	338	43.50	495	63.71
Technical	282	36.29	777	100.00

Figure 15.5

Department				
Department	**Frequency**	**Percent**	**Cumulative Frequency**	**Cumulative Percent**
Administration	157	20.21	157	20.21
Sales	338	43.50	495	63.71
Technical	282	36.29	777	100.00

Modifying the Default Font Sizes for RTF Output

When you want to change the font sizes for RTF output, you need to look at the default style definition for RTF output, which is Styles.RTF. Recall that Styles.RTF is based on Styles.Printer, which in turn is based on Styles.Default.

By running PROC TEMPLATE with a SOURCE statement for Styles.RTF, you can find out how Styles.RTF specifies the attributes in its fonts element. Figure 15.6 shows the fonts element in Styles.RTF.

To modify the fonts used for RTF output, you follow a similar process to that in the previous example by defining a new style that modifies the settings in the fonts element in Styles.RTF.

The following PROC TEMPLATE step creates template RTFSmaller that reduces the font sizes.

```
    PROC TEMPLATE;
  DEFINE STYLE RTFSmaller;
   PARENT=Styles.RTF;
   CLASS fonts /
    'TitleFont2' = ("Times",10pt,Bold Italic)
    'TitleFont' = ("Times",11pt,Bold Italic)
    'StrongFont' = ("Times",8pt,Bold)
    'EmphasisFont' = ("Times",8pt,Italic)
    'FixedEmphasisFont' = ("Courier New, Courier",7pt,Italic)
    'FixedStrongFont' = ("Courier New, Courier",7pt,Bold)
    'FixedHeadingFont' = ("Courier New, Courier",7pt,Bold)
    'BatchFixedFont' = ("SAS Monospace,Courier New,Courier",6pt)
    'FixedFont' = ("Courier New, Courier",7pt)
    'headingEmphasisFont' = ("Times",9pt,Bold Italic)
    'headingFont' = ("Times",9pt,Bold)
    'docFont' = ("Times",8pt);
  END;
RUN;
```

Compare Figure 15.7 to Figure 15.8. The PROC FREQ report on the data set HR in Figure 15.7 was created using Styles.RTF, whereas the report in Figure 15.8 was created using the new template RTFSmaller. Notice how the look of the two outputs is the same and that the font size has been scaled down in Figure 15.8.

Figure 15.6

```
define style Styles.Rtf;
  parent = styles.printer;

  style fonts /
     'TitleFont2' = ("<serif>, Times Roman",12pt,bold italic)
     'TitleFont' = ("<serif>, Times Roman",13pt,bold italic)
     'StrongFont' = ("<serif>, Times Roman",10pt,bold)
     'EmphasisFont' = ("<serif>, Times Roman",10pt,italic)
     'FixedEmphasisFont' = ("<monospace>, Courier",9pt,italic)
     'FixedStrongFont' = ("<monospace>, Courier",9pt,bold)
     'FixedHeadingFont' = ("<monospace>, Courier",9pt,bold)
     'BatchFixedFont' = ("SAS Monospace, <monospace>, Courier",6.7pt)
     'FixedFont' = ("<monospace>, Courier",9pt)
     'headingEmphasisFont' = ("<serif>, Times Roman",11pt,bold italic)
     'headingFont' = ("<serif>, Times Roman",11pt,bold)
     'docFont' = ("<serif>, Times Roman",10pt);
```

Figure 15.7

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Administration	157	20.21	157	20.21
Sales	338	43.50	495	63.71
Technical	282	36.29	777	100.00

Figure 15.8

Department				
Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Administration	157	20.21	157	20.21
Sales	338	43.50	495	63.71
Technical	282	36.29	777	100.00

The SAS 9.1.3 version of Styles.RTF does not include a fonts element. This is because in SAS 9.1.3, Styles.RTF inherits the fonts element from the Styles.Printer template. To accomplish the same effect in SAS 9.1.3, you would modify the font settings list defined in Styles.Printer.

Modifying the Default Typefaces

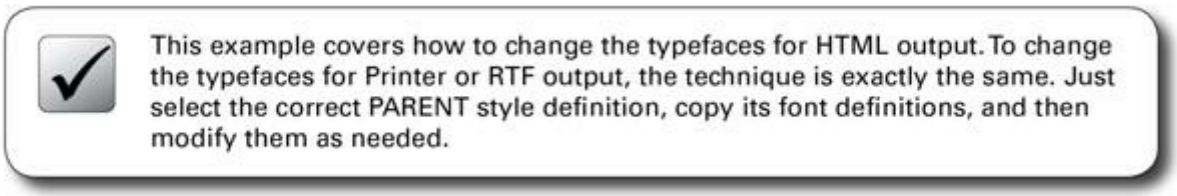
Modifying the typefaces used in your output is simple. Just as with the font sizes, all you have to do to change the typefaces is to copy the old font definitions into a PROC TEMPLATE step and then use a CLASS statement or REPLACE statement to change the font names to the ones you want.

The PROC TEMPLATE step that follows defines a new template that specifies a variety of fonts to give you an idea how they affect your output. It is applied to a PROC GLM analysis of the data set POSITIONS, and Figure 15.9 shows the results.

```
    PROC TEMPLATE;
  DEFINE STYLE DefaultNewFonts;
    PARENT=Styles.Default;
    CLASS fonts /
    'TitleFont2' = ("Algerian",4,Bold Italic)
    'TitleFont' = ("Brittanic Bold",5,Bold Italic)
    'StrongFont' = ("Comic Sans MS",4,Bold)
    'EmphasisFont' = ("Desdemona",3,Italic)
    'FixedEmphasisFont' = ("Lucida Typewriter",2,Italic)
    'FixedStrongFont' = ("Letter Gothic MT",2,Bold)
    'FixedHeadingFont' = ("QuickType Mono",2)
    'BatchFixedFont' = ("SAS Monospace",2)
    'FixedFont' = ("Courier New",2)
    'headingEmphasisFont' = ("Eras Demi ITC",4,Bold Italic)
    'headingFont' = ("Franklin Gothic Medium",4,Bold)
    'docFont' = ("Goudy Old Style",3);
  END;
RUN;
```

There are several important things to note about this example. First, when you are replacing fonts in a style definition, you need to pay attention to which fonts are supposed to be fixed-width fonts. Those fonts should only be replaced with other fixed-width fonts. In the original version of this style definition, which is Styles.Default, the five fixed-width font definitions were all set to Courier or SAS Monospace. Those are both good, reliable fixed-width fonts. Depending on your system, you might have other fixed-width fonts available.

Second, you can't list just any font. It needs to be a font available on your system, and, for HTML output, it needs to be a font commonly available to users who will view your Web page. The next example gets into this in more detail.



This example covers how to change the typefaces for HTML output. To change the typefaces for Printer or RTF output, the technique is exactly the same. Just select the correct PARENT style definition, copy its font definitions, and then modify them as needed.

Figure 15.9

Table of Contents

1. The GLM Procedure
   - Data
     - Class Levels
     - Number of Observations
   - Analysis of Variance
     - Salary
       - Overall ANOVA
       - Fit Statistics
       - Type I Model ANOVA
       - Type III Model ANOVA
   - Means
     - Grade
       - Means

THE GLM PROCEDURE

Class Level Information		
Class	Levels	Values
Grade	4	I II III IV
Position	5	Associate Manager Specialist Trainee VP

Number of Observations Read	777
Number of Observations Used	777

THE GLM PROCEDURE

DEPENDENT VARIABLE: SALARY ANNUAL COMPENSATION

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	7	1.9817939E12	283113411138	138.77	<.0001

Identifying Available Fonts

If you are creating HTML output, you can select any font that you believe will be available to people who view your Web page. If you want to use a slightly unusual font, you can list alternative fonts in the definitions so that viewers who do not have the first choice font will still be able to view your output properly. The default definition does this by listing <sans-serif> and Helvetica as the basic fonts. If none of the fonts listed in your definition is available, the viewer's browser substitutes a default font, which is generally not very attractive. Table 15.2 has a list of good fonts to use for HTML output.

If you are creating RTF output, again, you can select any font that you believe will be available to people opening your Word document.

For PDF output, you can select any typeface that is available in your system. To quickly view the available fonts, open a new document in Word and select the drop-down list of fonts from the toolbar (shown in Figure 15.10) or select **Format** and then **Font** from the menu bar. Alternatively, you can open the Control Panel in Windows and select **Fonts**.

If you plan to share your document with other users, you might want to limit yourself to the more common fonts shown in Table 15.2.

If you are creating Printer output on Windows, you can use the default host printer drivers. This means that you have access to any font on your system. To get a list of these fonts, open your word processor and look up the font list using the approach described previously.

For Printer output on any other platform, you cannot use host drivers. By default, you are using the SAS drivers. SAS ships with an extensive list of fonts, including three main font families: Albany AMT (Arial), Thorndale AMT (Times New Roman), and Cumberland AMT (Courier New). Also included are two symbol fonts and eight Asian language fonts.

Whichever operating system you are using, if you're not sure which fonts your printer supports, the best thing to do is experiment. Create a series of style definitions that use a variety of fonts. Generate some test output and review the results. The fonts that are not supported on your system or printer will be displayed as one of the standard fonts (probably Courier). Start a list of the fonts that work and don't work and use it as a reference when you are building style definitions.

Unless you have added the SAS or PS option to your ODS PRINTER statement, you are using the host drivers. If you are on the Windows platform, it is recommended that you not change this setting unless you are having problems printing.

To ensure that your SAS system knows all about the fonts installed on your computer, you need to run PROC FONTREG. The FONTREG procedure enables you to update the SAS registry to include system fonts, which can then be used in SAS output. The following code shows the syntax for Windows (modify the path for your system):

```
PROC FONTREG MODE=ALL;
   FONTPATH 'c:\winnt\fonts';
RUN;
```

Figure 15.10

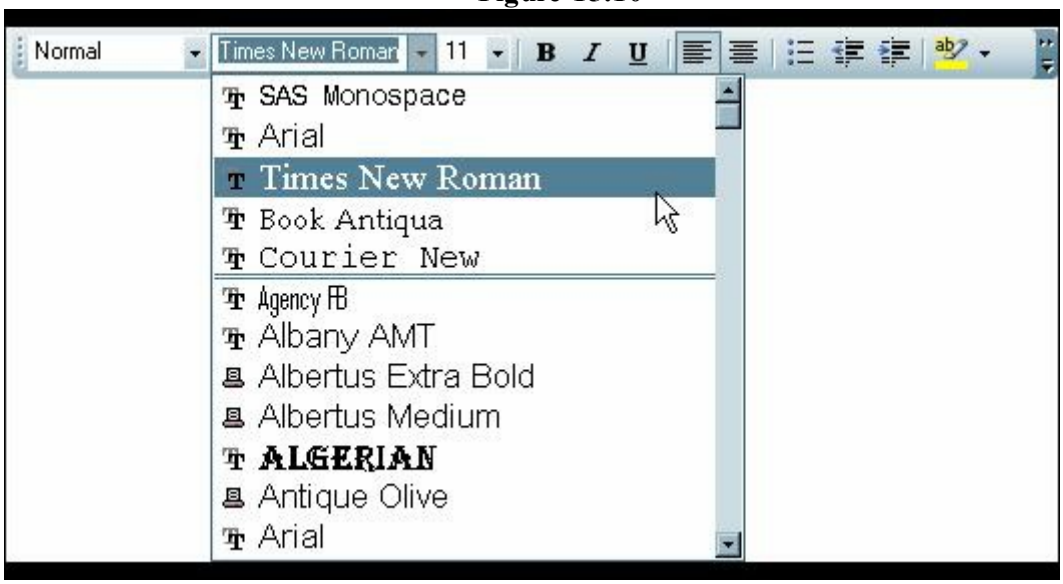


Table 15.2

Widely available fonts to use on the Internet[1]	Rare fonts[2]
Times New Roman, Times	SAS Monospace, Courier
Arial, Helvetica	**SAS Monospace Bold,** Courier
Courier New, Courier	
Comic Sans MS, Arial, Helvetica	
Verdana, Arial, Helvetica	
Impact, Arial Black, Helvetica	
Georgia, Times New Roman, Times	
Trebuchet MS, Arial, Helvetica	

Modifying the Default Font Weights, Font Styles, and Font Widths

The previous examples have looked at how to modify font typefaces and sizes. These are the first two items listed in each font definition. The third item is a list of keywords that defines the font weight (bold3_, for instance), the font style (italic, for instance), and the font width (compressed, for instance). These font weights and styles can be modified just as you modified the typefaces and sizes.

The RTF style definition uses four different combinations of font weight, font style, and font width: none, bold only, italic only, and bold italic. To see how to change these font definitions, we will change the style element so that it does not use italics. All we have to do is delete the word "Italic" from each definition. Figure 15.11 shows the original font definitions for the RTF style definition. The revised fonts style element with the new font definitions is shown below in the PROC TEMPLATE step.

```
    PROC TEMPLATE;
  DEFINE STYLE RTFNoItalics;
    PARENT=Styles.RTF;
  CLASS fonts /
    'TitleFont2' = ("Times",12pt,Bold)
    'TitleFont' = ("Times",13pt,Bold)
    'StrongFont' = ("Times",10pt,Bold)
    'EmphasisFont' = ("Times",10pt)
    'FixedEmphasisFont' = ("Courier New, Courier",9pt)
    'FixedStrongFont' = ("Courier New, Courier",9pt,Bold)
    'FixedHeadingFont' = ("Courier New, Courier",9pt,Bold)
    'BatchFixedFont' = ("SAS Monospace,Courier New,Courier",6.7pt)
    'FixedFont' = ("Courier New, Courier",9pt)
    'headingEmphasisFont' = ("Times",11pt,Bold)
    'headingFont' = ("Times",11pt,Bold)
    'docFont' = ("Times",10pt);
  END;
RUN;
```

To see the impact of these changes, compare Figure 15.12 and Figure 15.13. In these examples, you can see that the two titles, which were italicized in Figure 15.12, are not italicized in Figure 15.13. For this simple PROC MEANS step on the data set POSITIONS, these titles are the only place where italics were applied. In a more complex example, these changes would have affected other headings, and also any emphasized results.

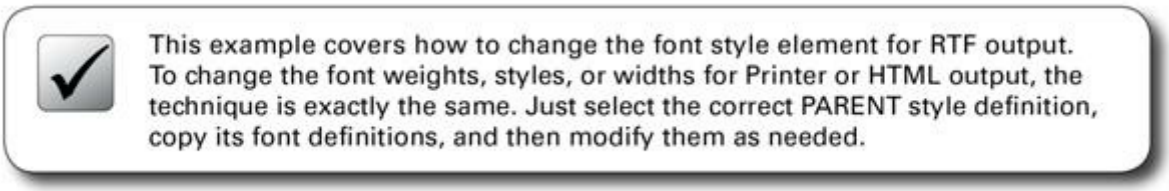


Figure 15.11

```
style fonts /
   'TitleFont2' = ("<serif>, Times Roman",12pt,bold italic)
   'TitleFont' = ("<serif>, Times Roman",13pt,bold italic)
   'StrongFont' = ("<serif>, Times Roman",10pt,bold)
   'EmphasisFont' = ("<serif>, Times Roman",10pt,italic)
   'FixedEmphasisFont' = ("<monospace>, Courier",9pt,italic)
   'FixedStrongFont' = ("<monospace>, Courier",9pt,bold)
   'FixedHeadingFont' = ("<monospace>, Courier",9pt,bold)
   'BatchFixedFont' = ("SAS Monospace, <monospace>, Courier",6.7pt)
   'FixedFont' = ("<monospace>, Courier",9pt)
   'headingEmphasisFont' = ("<serif>, Times Roman",11pt,bold italic)
   'headingFont' = ("<serif>, Times Roman",11pt,bold)
   'docFont' = ("<serif>, Times Roman",10pt);
```

Figure 15.12

Frequencies in Data Set POSITIONS

The FREQ Procedure

Experience Level				
Grade	Frequency	Percent	Cumulative Frequency	Cumulative Percent
I	273	35.14	273	35.14
II	256	32.95	529	68.08
III	132	16.99	661	85.07
IV	116	14.93	777	100.00

Figure 15.13

Frequencies in Data Set POSITIONS

The FREQ Procedure

Experience Level				
Grade	Frequency	Percent	Cumulative Frequency	Cumulative Percent
I	273	35.14	273	35.14
II	256	32.95	529	68.08
III	132	16.99	661	85.07
IV	116	14.93	777	100.00

Creating a New Font Definition

The easiest way to change font definitions is to just edit the existing font definitions. However, some times you might prefer more flexibility.

For example, if you want to assign a new font to specific parts of your output, you can't do that with the existing font definitions. In the default style template, the same font attributes are used for both titles and footnotes. So if you want to write the footnotes in a small, unobtrusive font, you have to also use the same font for the titles.

There are a couple of ways to create a style template with large bold titles and small subtle footnotes. One way is to create a new version of the fonts style element. The PROC TEMPLATE step that follows includes a new attribute, FOOTNOTEFONT=, in the fonts element. Note that the step is not complete until the statement near the bottom of the page is added.

```
    PROC TEMPLATE;
  DEFINE STYLE DefaultNewFonts2;
   PARENT=Styles.Default;
   CLASS fonts /
    'TitleFont2' = ("Arial, Helvetica, Helv",4,Bold Italic)
    'TitleFont' = ("Arial, Helvetica, Helv",5,Bold Italic)
    'StrongFont' = ("Arial, Helvetica, Helv",4,Bold)
    'EmphasisFont' = ("Arial, Helvetica, Helv",3,Italic)
    'FixedEmphasisFont' = ("Courier",2,Italic)
    'FixedStrongFont' = ("Courier",2,Bold)
    'FixedHeadingFont' = ("Courier",2)
    'BatchFixedFont' = ("SAS Monospace, Courier",2)
    'FixedFont' = ("Courier",2)
    'headingEmphasisFont' = ("Arial,Helvetica,Helv",4,Bold Italic)
    'headingFont' = ("Arial, Helvetica, Helv",4,Bold)
    'docFont' = ("Arial, Helvetica, Helv",3)
    'FootnoteFont'= ("Arial, Helvetica, Helv", 1, Italic);
```

Next, the new font definition has to be applied to the footnotes. To do this, we need to run a PROC TEMPLATE step with a SOURCE statement for Styles.Default to find the part of the code that sets the font for footnotes. Figure 15.14 shows the code. To use our new font definition, we just replace the original font definition of TitleFont with our new font definition FootnoteFont.

```
      CLASS SystemFooter /
    font = Fonts('FootnoteFont');
  END;
RUN;
```

After we create the style template DefaultNewFonts2, we can now create output like Figure 15.15, which has large titles and tiny footnotes. You can use this technique any time you want to control the fonts used for a particular part of the output. Just because the default style definition reuses the same fonts for a number of output sections doesn't limit you to using those fonts. Just create a new font definition, and then apply it to the part of the output that you want to change.

Figure 15.14

```
class SystemFooter /
   font = Fonts('TitleFont');
```

Figure 15.15

Salary Statistics for Each Department

The MEANS Procedure

Analysis Variable : Salary Annual Compensation						
Department	N Obs	N	Mean	Minimum	Median	Maximum
Administration	157	157	78648.43	23831.33	55642.67	372137.40
Sales	338	338	88812.35	20922.58	69052.83	367651.80
Technical	282	282	81620.26	21117.67	65062.83	491465.00

Data Set: POSITIONS
Data Set Last Updated: 30JUN2010

If you want to change the font in only one place, and don't plan to use that font style anywhere else, you can create a new font definition inside the SystemFooter style element, and avoid changing the fonts style element. Here is the syntax:

```
CLASS SystemFooter /
  font=("Arial, Helvetica, Helv", 1, Italic);
```

Modifying Fonts for Procedures That Use Fixed-Width Fonts

In each of the previous examples, we've been ignoring the definitions for the fixed-width fonts. For most SAS procedures, these fonts are never used. However, there are a few procedures where these fonts matter. For example, the PLOT, CHART, TIMEPLOT, and CALENDAR procedures all use fixed-width fonts to get their output to line up properly. It is possible to modify the style attributes for these fonts. However, do this carefully, and review your results. Because these fonts are so integral to the appearance of these procedures' results, changes to the font definitions can cause problems with the appearance of the results.

If you are creating HTML output, be aware that the output from these procedures might not look exactly correct even if you use the Default style definition. Some parts of the output of these procedures use the SAS Monospace font. If this font is not available on your browser, it will default to Courier. The problem with that is that the output uses some special characters available in the SAS Monospace font. When your browser substitutes Courier, you end up with strange characters in your output. To fix the problem, use the FORMCHAR option to specify safer characters:

```
OPTIONS FORMCHAR="|----|+|---+=|-/\<>*";
```

If you are creating Printer output, be wary about changing the font definitions. If you make the font sizes larger, your output might not line up. In addition, if you change typefaces, you might get strange characters in your output because the procedure was expecting the SAS Monospace font. The only settings you can safely change are the font weight and style. You can switch from regular to bold, italic to roman, etc.

With RTF output, changing the typeface to something other than SAS Monospace will cause problems with the alignment of your output, but you can safely change the font sizes and weights. This example code changes the font size and weight for BATCHFIXEDFONT=, which is the font that is used for PROC CHART output.

```
    PROC TEMPLATE;
  DEFINE STYLE RTFFixed;
   PARENT=Styles.RTF;
  REPLACE fonts /
   'TitleFont2' = ("Times",12pt,Bold Italic)
   'TitleFont' = ("Times",13pt,Bold Italic)
   'StrongFont' = ("Times",10pt,Bold)
   'EmphasisFont' = ("Times",10pt,Italic)
   'FixedEmphasisFont' = ("Courier New, Courier",9pt,Italic)
   'FixedStrongFont' = ("Courier New, Courier",9pt,Bold)
   'FixedHeadingFont' = ("Courier New, Courier",9pt,Bold)
   'BatchFixedFont' = ("SAS Monospace,Courier New,Courier",12pt, Bold)
   'FixedFont' = ("Courier New, Courier",9pt)
   'headingEmphasisFont' = ("Times",11pt,Bold Italic)
   'headingFont' = ("Times",11pt,Bold)
   'docFont' = ("Times",10pt);
  END;
 RUN;
```

By using this font, you can make the bar chart stand out a bit more. Compare the

readability of Figure 15.16, which uses a regular 9-point font, with Figure 15.17, which uses a bold 12-point font.

Figure 15.16

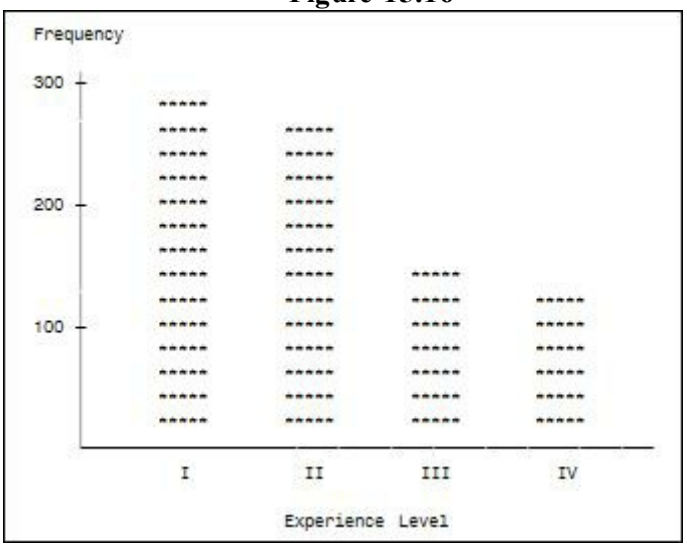
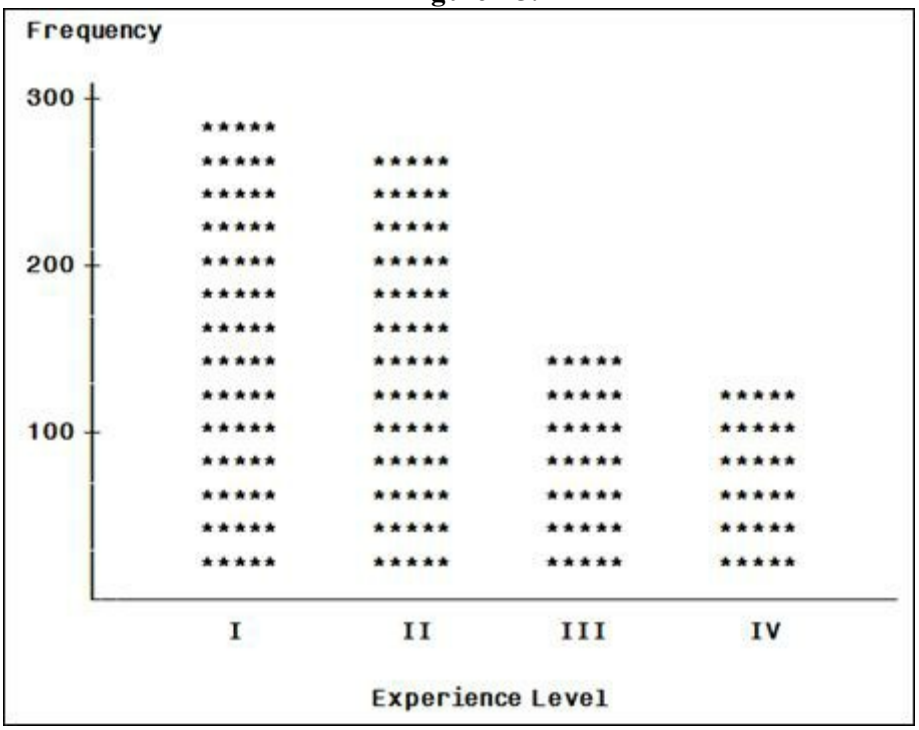


Figure 15.17



Chapter 16
Modifying Output Structure

The previous chapter was just the beginning. Modifying the fonts gives you control over all of the text portions of your output. However, surrounding all of that text are things like tables, borders, bullets, and headings.

ODS enables you to change each of these items. You can change the shape of a table or modify its borders. You can change or remove the bullets in a table of contents or pages. You can change page and procedure headings.

The possibilities are infinite. This chapter gives you some of the basic code building blocks you need to make these kinds of changes. After that, it's up to you. You can reuse and combine these bits of code to create the look you desire.

In our upcoming examples, we feature SAS 9.2 syntax. The author pages for this book, at http://support.sas.com/authors, contain SAS 9.1.3 versions of the style template programs.

This chapter uses the data set BUILDINGS to illustrate the concepts. This data set is also available on the author pages for this book.

Modifying the Table Borders

Most ODS output is rendered in the form of tables. This allows your output to be organized and easy to read. The appearance of these tables is controlled by the style definition.

To see how different table designs can affect your output, just compare the HTML output in Figure 16.1 to that in Figure 16.2. Figure 16.1 shows how the Styles.Default style template formatted the PROC MEANS report on the data set BUILDINGS. The table borders in Figure 16.1 are flat-looking and narrow. Figure 16.2 shows how the style template Styles.D3D distributed with SAS formats the same report. The modified attributes for the table borders in the style template D3D produce wide borders that use shading to achieve a three-dimensional appearance in the HTML destination.

You can change the borders on your output tables by specifying a different style template with the STYLE= option on your ODS statement, or you can create a new style template with code that changes the table borders. For example, if you like the general appearance that the default style template produces, but you also like the wider borders produced by the D3D style template, you can create a new style template that combines these characteristics.

The first step in modifying the borders is to find the code that formats the borders in the style template you want to adapt. Start by running a PROC TEMPLATE step with a SOURCE statement for the style template. From the output, identify the element in the style template that controls the border widths. It turns out that there are a number of border settings. The Output style element is where the settings for the main table borders are located in Styles.Default. Figure 16.3 shows the original attributes of this style element.

To make the border wider, all you have to do is change the BORDERWIDTH= attribute setting. The following PROC TEMPLATE step creates a new style template called WiderBorder, which has a border width setting of 10.

```
    PROC TEMPLATE;
  DEFINE STYLE WiderBorder;
    PARENT=Styles.Default;
    CLASS Output /
            bordercollapse=separate
      background = colors('tablebg')
      rules = GROUPS
      frame = BOX
      cellpadding = 7
      cellspacing = 1
      bordercolor = colors('tableborder')
      borderwidth = 10;
  END;
RUN;
```

Figure 16.4 shows the results of applying the style template WiderBorder to the same PROC MEANS report shown in Figure 16.1 and Figure 16.2.

To accomplish the same border change in SAS 9.1.3, switch the STYLE statement to a REPLACE statement, remove the BORDERCOLLAPSE= attribute, and add "**from Container**" after the style element name Output.

Figure 16.1

Analysis Variable : CostSqFt Cost per square foot					
Building type	N	Mean	Std Dev	Minimum	Maximum
Office	135	300.12	56.19	202.00	400.00
Residential	118	291.75	56.46	201.00	398.00
Retail	29	304.62	60.71	204.00	400.00
Warehouse	60	297.97	57.68	202.00	398.00

Figure 16.2

Analysis Variable : CostSqFt Cost per square foot					
Building type	N	Mean	Std Dev	Minimum	Maximum
Office	135	300.12	56.19	202.00	400.00
Residential	118	291.75	56.46	201.00	398.00
Retail	29	304.62	60.71	204.00	400.00
Warehouse	60	297.97	57.68	202.00	398.00

Figure 16.3

```
class Output /
   bordercollapse = separate
   backgroundcolor = colors('tablebg')
   rules = GROUPS
   frame = BOX
   padding = 7
   borderspacing = 1
   bordercolor = colors('tableborder')
   borderwidth = 1;
```

Figure 16.4

Analysis Variable : CostSqFt Cost per square foot					
Building type	N	Mean	Std Dev	Minimum	Maximum
Office	135	300.12	56.19	202.00	400.00
Residential	118	291.75	56.46	201.00	398.00
Retail	29	304.62	60.71	204.00	400.00
Warehouse	60	297.97	57.68	202.00	398.00

Removing the Table Borders and Rules

The previous example defined a style template for tables with wider borders. You can also make the borders and rules go away altogether. All you have to do is modify a few other settings in the style element we worked with in the previous example.

We are working again with the Output style element in the default style template, Styles.Default. As a reminder, the original settings of the Output style element are shown on the previous page in Figure 16.3.

Three settings in Output need to be changed if we want to create a table without borders or rules. The first one is the RULES= attribute. The setting for RULES= in Styles.Default is GROUPS. This calls for rules between the table header and the table body, and between the table body and the table footer. If we want to get rid of these rules, the setting must be changed to NONE. Figure 16.5 shows the revised HTML output with this setting in place.

The second setting that is used to control borders is the frame attribute setting. The value for this setting in Styles.Default is FRAME=BOX. This setting calls for borders around the outside of the table. To get rid of these borders, change the setting for FRAME= to VOID. Figure 16.6 shows what the table borders look like with this second change in place.

At this point, you might think that the task is complete. However, if you look closely at Figure 16.6, you will see thin white rules between the table cells. These are not actually rules, but are spaces between the table cells. However, they look like rules, so we'd like to remove them. To do this we need to change the CELLSPACING= attribute setting from 1 to 0. Our final code is shown below. It creates Figure 16.7.

```
    PROC TEMPLATE;
  DEFINE STYLE DefaultNoBorderRule;
   PARENT=Styles.Default;
   CLASS Output /
      background = colors('tablebg')
      rules = NONE
      frame = VOID
      cellpadding = 7
      cellspacing = 0
      bordercolor = colors('tableborder')
      borderwidth = 1;
  END;
RUN;
```

Figure 16.5

Analysis Variable : CostSqFt Cost per square foot					
Building type	N	Mean	Std Dev	Minimum	Maximum
Office	135	300.12	56.19	202.00	400.00
Residential	118	291.75	56.46	201.00	398.00
Retail	29	304.62	60.71	204.00	400.00
Warehouse	60	297.97	57.68	202.00	398.00

Figure 16.6

Analysis Variable : CostSqFt Cost per square foot					
Building type	N	Mean	Std Dev	Minimum	Maximum
Office	135	300.12	56.19	202.00	400.00
Residential	118	291.75	56.46	201.00	398.00
Retail	29	304.62	60.71	204.00	400.00
Warehouse	60	297.97	57.68	202.00	398.00

Figure 16.7

Analysis Variable : CostSqFt Cost per square foot					
Building type	N	Mean	Std Dev	Minimum	Maximum
Office	135	300.12	56.19	202.00	400.00
Residential	118	291.75	56.46	201.00	398.00
Retail	29	304.62	60.71	204.00	400.00
Warehouse	60	297.97	57.68	202.00	398.00

You might have noticed that in the final code, the BORDERWIDTH= setting was still set to 1. Because we used the FRAME= and RULES= attributes to get rid of all borders, it does not matter that the width is still set to 1, since no borders will be created.

Removing Table Borders and Rules from Printer Output

So far in this chapter, the examples have been based on the Styles.Default definition, and the output destination has been HTML. This time we will work with the default style template definition for the PRINTER destination, Styles.Printer, and create printed output. Again, we will be removing the table borders and rules. In the following code, you can see that the modifications are quite different from the previous example. First, the parent style definition is now listed as Styles.Printer. Second, instead of replacing the Output style element, we replace the Table style element. Figure 16.8 shows the source code with the attributes and settings for the Table style element.

Why do we have to replace a different style element? In Styles.Default, the settings for RULES=, FRAME=, and CELLSPACING= come from the Output style element. In Styles.Printer, the RULES= setting in the Output style element is overruled by the RULES= setting in the Table style element. Similarly, the CELLSPACING= setting in the Output style element is overruled by the CELLSPACING= setting in the Table style element. The FRAME= setting still comes from the Output style element (see the following tip for an explanation).

In the following PROC TEMPLATE step, the modifications to the Table style element are the same as the modifications to the Output style element in the previous example. The RULES= attribute is set to NONE, FRAME= is set to VOID, and CELLSPACING= is set to zero. The FRAME= attribute was not part of the original Table style element we are updating, but it's a valid attribute for that style element.

If a style element you are modifying does not have all of the attributes you want to modify, you can often add them. Just check the ODS manual in the SAS documentation for that attribute to be sure it is valid for the style element you are trying to modify. However, if you accidentally specify an invalid attribute, it won't cause an error. ODS simply ignores extra attributes.

```
    PROC TEMPLATE;
  DEFINE STYLE PrinterNoBorder;
    PARENT=Styles.Printer;
    CLASS Table /
      rules = NONE
      frame = VOID
      cellpadding = 4pt
      cellspacing = 0
      borderwidth = 1;
  END;
RUN;
```

Figure 16.9 shows a sample table using the Styles.Printer style template. Figure 16.10 shows the same table again after applying the new PrinterNoBorder style template.

This example might be a little confusing in SAS 9.1.3, where the Styles.Printer style template doesn't have an Output style element, yet the Table style element refers to it. The Output style element is in Styles.Default. Our new style template can replace style elements in either Styles.Printer or in its parent, Styles.Default.

Figure 16.8: Default Settings for the Table Style Element

```
style Table from Output /
   bordercollapse = separate
   rules = ALL
   padding = 4pt
   borderspacing = 0.25pt
   borderwidth = 0.75pt;
```

Figure 16.9

Analysis Variable : CostSqFt Cost per square foot					
Building type	N	Mean	Std Dev	Minimum	Maximum
Office	135	300.12	56.19	202.00	400.00
Residential	118	291.75	56.46	201.00	398.00
Retail	29	304.62	60.71	204.00	400.00
Warehouse	60	297.97	57.68	202.00	398.00

Figure 16.10

Analysis Variable : CostSqFt Cost per square foot					
Building type	N	Mean	Std Dev	Minimum	Maximum
Office	135	300.12	56.19	202.00	400.00
Residential	118	291.75	56.46	201.00	398.00
Retail	29	304.62	60.71	204.00	400.00
Warehouse	60	297.97	57.68	202.00	398.00

Cell borders can be modified individually as well by using inline style functions. You can assign a color or width to each cell border: top, bottom, left, and right. These values can be changed at the single-cell level, allowing complete control over table borders.

Changing the Spacing Inside Tables

ODS output is composed of tables, text, and the space between them. By default, a certain amount of space is left between the results and the table borders that surround them. If you'd like to create a more compressed table with less space, or a more open table with more space, you can define a style template to do that.

The attribute setting you need to modify to change this spacing is CELLPADDING=. Like the table border settings, it's part of the Output style element of Styles.Default. The default setting is 7 pixels. This means that to the left and right of each number or label in the table, there are at least 7 pixels of space. There are also 7 pixels of space above and below each number or label.

If you want to create a more compressed table, you simply reduce the value of the CELLPADDING= setting. The following code sets the CELLPADDING= to 3 pixels.

```
    PROC TEMPLATE;
  DEFINE STYLE DefaultSqueeze;
   PARENT=Styles.Default;
   CLASS Output /
      background = colors('tablebg')
      rules = GROUPS
      frame = BOX
      cellpadding = 3
      cellspacing = 1
      bordercolor = colors('tableborder')
      borderwidth = 1;
  END;
RUN;
```

The results of this change can be seen by comparing Figures 16.11 and 16.12. The HTML output in Figure 16.11 was created with Styles.Default, whereas the report in Figure 16.12 was created with the new style template DefaultSqueeze.

To see the opposite effect, look at Figure 16.13. It was created using the same code as above except for the following line:

```
cellpadding = 10
```

Looking at the column for the N statistic, you can see how much more space is allowed in this table. It now has 10 pixels of space around each number and label.

In general, you will find that the default setting is just fine. However, if you need to squeeze a bit more output on a page, reducing the cell padding is one way to go. Similarly, if you want to leave a bit more room to make a crowded table easier to read, increase the cell padding.

This example is for HTML output. If you are creating RTF or Printer output, the CELLPADDING= attribute uses settings in points. The default value is 4pt for Printer output and 3pt for RTF.

Figure 16.11: Table with default CELLPADDING=7

Analysis Variable : CostSqFt Cost per square foot					
Building height	N	Mean	Std Dev	Minimum	Maximum
1 story	117	290.75	58.34	201.00	398.00
2 stories	122	300.83	55.87	202.00	400.00
high-rise	49	295.59	54.70	202.00	400.00
mid-rise	54	304.65	57.63	202.00	400.00

Figure 16.12: Table with CELLPADDING=3

Analysis Variable : CostSqFt Cost per square foot					
Building height	N	Mean	Std Dev	Minimum	Maximum
1 story	117	290.75	58.34	201.00	398.00
2 stories	122	300.83	55.87	202.00	400.00
high-rise	49	295.59	54.70	202.00	400.00
mid-rise	54	304.65	57.63	202.00	400.00

Figure 16.13: Table with CELLPADDING=10

Analysis Variable : CostSqFt Cost per square foot					
Building height	N	Mean	Std Dev	Minimum	Maximum
1 story	117	290.75	58.34	201.00	398.00
2 stories	122	300.83	55.87	202.00	400.00
high-rise	49	295.59	54.70	202.00	400.00
mid-rise	54	304.65	57.63	202.00	400.00

Removing Numbers from the Table of Contents in HTML Output

Back in Chapter 3, "HTML Output," there was an example that showed how to remove the procedure labels from an HTML table of contents. By using the ODS PROCLABEL statement, we were able to remove the text labels such as "The MEANS Procedure" and "The PRINT Procedure" from the table of contents listing.

Using a blank for the PROCLABEL setting (ODS PROCLABEL ' '), you can create the Web page shown in Figure 16.14. The procedure labels have now been removed. The problem is that although the unwanted text in the table of contents is gone, the number for that item is still in place. The "1." is still displayed. To get rid of the "1.", you can't use a simple ODS command. This functionality is controlled by the style template definition. To remove the "1.", you have to create a new style template that does not call for the item numbers.

For many style definition modifications, you can run PROC TEMPLATE with a SOURCE statement to get the source code listing of the style template you want to adapt. Then you can read through the code and identify the style element and attributes that control the item you want to change. However, if you read through the Styles.Default template, you will quickly discover that there is no attribute called TableOfContentsNumbers or anything like that.

This is where SAS ODS online documentation comes in handy. Each of the style elements, style attributes, and their settings are explained in the chapter on the TEMPLATE procedure.

Here is how you use the documentation to find what you need. Open the ODS documentation on support.sas.com, and search for "ODS Style Elements." This should bring up a page titled "General ODS Style Elements." Reading this table, you will discover that the ContentProcName style element controls the procedure labels in the table of contents.

If you look at the source code for Styles.Default, you will not find the ContentProcName style element, but you will find it in Base.Template.Style on which Styles.Default is based. You will see that the ContentProcName style element has no attributes. However, the code in Base.Template.Style indicates that it comes from the IndexProcName style element. This element is also in Styles.Default. If you look at the IndexProcName style element in Styles.Default, you see a number of attributes. There's no setting that is obviously the one we are looking for, but BULLET= "decimal" is a possibility. To be sure this is the right setting, you could run a test, or you could also look at the Index style element (on which IndexProcName is based) and the Container style element (on which Index is based). Neither of these style elements have anything more promising, so the BULLET= attribute is probably what we need.

Now you can look up that style attribute in the documentation and learn how to change its value. It turns out that NONE is the setting that removes the bullet. Here is the code to solve the problem:

```
PROC TEMPLATE;
DEFINE STYLE NoBullet;
 PARENT=STYLES.DEFAULT;
```

```
    CLASS ContentProcName /
          BULLET=NONE;
  END;
RUN;
```

This code creates the Web page in Figure 16.15. Notice that the "1." is no longer displayed. There's still a blank line, but at least the orphan number is no longer there.

Figure 16.14

Table of Contents

1.

-Table Material

-One-Way Frequencies

The FREQ Procedure

Primary construction material				
Material	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Brick	76	22.22	76	22.22
Concrete	115	33.63	191	55.85
Steel	47	13.74	238	69.59
Wood	104	30.41	342	100.00

Figure 16.15

Table of Contents

-Table Material

-One-Way Frequencies

The FREQ Procedure

Primary construction material				
Material	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Brick	76	22.22	76	22.22
Concrete	115	33.63	191	55.85
Steel	47	13.74	238	69.59
Wood	104	30.41	342	100.00

Setting the Width for HTML Output

The challenge to creating output for the Web is that the people who view your results will be using a variety of browsers on a variety of screen sizes and resolutions. HTML is a great tool for handling this, as it can adjust the width of various objects to the width of the browser window. This automatic behavior is great for fitting output on the screen, but what it means is that your results will look different depending on the size of the browser window. For example, in Figure 16.16, the table of contents is too narrow and some of the labels wrap awkwardly because the browser window was too small. This might be fine for your purposes. However, if you would like your output to look the same wherever it is viewed, there are some settings you can change.

This example shows how to set a fixed width for the table of contents and for the tables in the body of your output. Together, these changes will enforce a standard look for your output. We assign an overall width of 600 pixels to our results. The first 200 pixels are used for the table of contents, and the remaining 400 are used by the output tables. These results should be viewable by anyone with at least 640x480 screen resolution if they maximize the browser window.

We work with the Styles.Default template for this example. There are two settings we need to change. First, we need to add a width setting to our output tables. This is done in the Table style element. If you look at that style element in the source code, you will see that it has no attributes. However, a review of the documentation[1] reveals that SAS has an OUTPUTWIDTH= attribute that you can use. We will set the output width to 400. In SAS 9.2, the WIDTH= option is equivalent to OUTPUTWIDTH=, and you can use either.

We also need to fix the width of the table of contents. This width is controlled by the Frame style element. The attribute contentsize can be set to a percentage of the screen or to a fixed width. We will set it to a fixed width of 200 pixels.

```
    PROC TEMPLATE;
  DEFINE STYLE FixedWidth;
   PARENT=STYLES.DEFAULT;
   CLASS Table / outputwidth=400;
   CLASS Frame /
      contentposition = L
      bodyscrollbar = auto
      bodysize = *
      contentscrollbar = auto
      contentsize = 200
      framespacing = 1
      frameborderwidth = 4
      frameborder = on;
  END;
 RUN;
```

The revised output is shown in Figure 16.17. Now the table of contents has enough room that the labels do not wrap so awkwardly, and there is still enough space for the tables. If the browser window were any smaller, however, you would have to use the scroll bar to move

over to view the right side of the tables.

Figure 16.16

Table of Contents

1. The Univariate Procedure
   - CostSqFt
     - Moments
     - Basic Measures of Location and Variability
     - Tests For Location
     - Quantiles
     - Extreme Observations

The UNIVARIATE Procedure
Variable: CostSqFt (Cost per square foot)

Moments			
N	342	Sum Weights	342
Mean	297.233918	Sum Observations	101654
Std Deviation	56.8410163	Variance	3230.90113
Skewness	0.08618204	Kurtosis	-1.13331
Uncorrected SS	31316754	Corrected SS	1101737.29
Coeff Variation	19.1233277	Std Error Mean	3.07361015

Basic Statistical Measures	
Location	Variability

Figure 16.17

Table of Contents

1. The Univariate Procedure
   - CostSqFt
     - Moments
     - Basic Measures of Location and Variability
     - Tests For Location
     - Quantiles
     - Extreme Observations

The UNIVARIATE Procedure
Variable: CostSqFt (Cost per square foot)

Moments			
N	342	Sum Weights	342
Mean	297.233918	Sum Observations	101654
Std Deviation	56.8410163	Variance	3230.90113
Skewness	0.08618204	Kurtosis	-1.13331
Uncorrected SS	31316754	Corrected SS	1101737.29
Coeff Variation	19.1233277	Std Error Mean	3.07361015

Controlling RTF Page Margins

When you create a word processor document using ODS RTF, you might have noticed that when you open the document, it has very narrow margins. By default, ODS creates RTF output with one-quarter inch margins on the top, bottom, right, and left sides. You can see these margins by opening an RTF file created by ODS and selecting **Page Setup**. Figure 16.18 shows the margin settings from the Page Setup window in Microsoft Word 2003. All four margins are set to 0.25", which is the default value for ODS RTF output.

Starting in SAS 9.2, it's easy to adjust the margins by changing the SAS system options for margins: TOPMARGIN=, BOTTOMMARGIN=, LEFTMARGIN=, and RIGHTMARGIN=. For example, the following statement preceding your ODS RTF statement changes the four margins to 1 inch.

```
OPTIONS TOPMARGIN=1in BOTTOMMARGIN=1in LEFTMARGIN=1in
      RIGHTMARGIN=1in;
```

In earlier SAS versions, the RTF output used the style template margin settings of 0.25 inches for all four margins. In order to change the margins, you needed to change the style.

The code that follows creates a style with new margin definitions that you can use in earlier versions of SAS. It copies all other settings from the ODS RTF style by specifying the PARENT=STYLES.RTF statement in the PROC TEMPLATE step. After creating the style, add to your ODS RTF statement the STYLE= option that specifies the name of your new style.

```
    PROC TEMPLATE;
  DEFINE STYLE RtfMargins;
    PARENT = Styles.RTF;
    STYLE BODY FROM BODY /
      BOTTOMMARGIN = .5in TOPMARGIN = .5in
      RIGHTMARGIN = .5in  LEFTMARGIN = .5in;
  END;
RUN;
```

If you run your ODS RTF output using this new style, your word processor document will have 0.5 inch margins, as shown in Figure 16.19.

However, if it was necessary to create a new style every time we wanted to change margins, this could become a nuisance. Instead of changing the style every time, you can create a single style for RTF output that has undefined margins. Then you can use the system option settings to define your margins as needed.

```
    PROC TEMPLATE;
  DEFINE STYLE RtfMargins;
    PARENT = Styles.RTF;
    STYLE BODY FROM BODY /
      BOTTOMMARGIN = _undef_  TOPMARGIN = _undef_
      RIGHTMARGIN = _undef_   LEFTMARGIN = _undef_;
  END;
RUN;
OPTIONS LEFTMARGIN=1in RIGHTMARGIN=1in TOPMARGIN=1.5in
      BOTTOMMARGIN=1in;
```

The page setup results for this combination of style and options are shown in Figure 16.20.

Figure 16.18

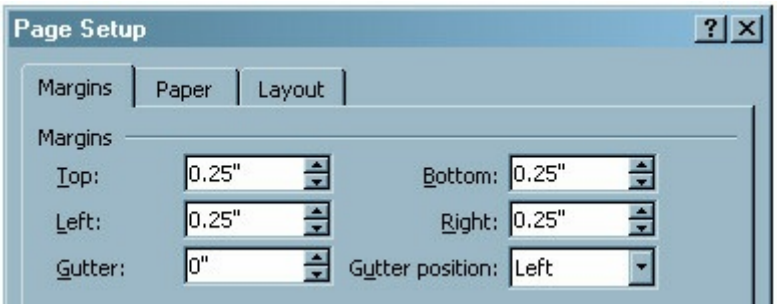


Figure 16.19

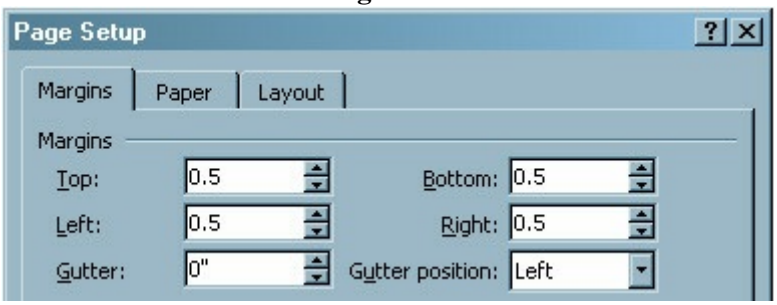
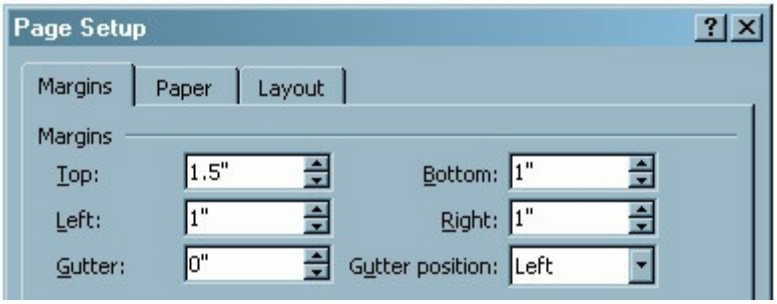


Figure 16.20



Simulating Listing Output with FILE PRINT ODS

What if you really like the way the listing output looks? Some people ask whether their ODS output can look like the output sent to the LISTING destination. Generally speaking, the use of ODS style templates is what gives each destination the particular look and feel that it has. If you create a custom style template and modify all the elements, such as proportional fonts and colors from the style, then you can come close to the look and feel of listing output. For example, the following PROC TEMPLATE step creates a style called LOOKLIST. It changes all the fonts for all the font attributes to Courier 8pt, which is a fixed pitch, or monospace, font.

The ODS PATH statement causes PROC TEMPLATE to save the style template LOOKLIST in an item store entitled WORK.TEMPLAT, which is in the WORK directory. Template LOOKLIST is stored in a folder called STYLES in the WORK.TEMPLAT directory. The second part of the statement causes PROC TEMPLATE to read Styles.Printer from SASHELP.TMPLMST.

```
ODS PATH WORK.TEMPLAT(UPDATE) SASHELP.TMPLMST(READ);
PROC TEMPLATE;
  define style styles.looklist;
   parent=styles.printer;
    style fonts from fonts /
     'BatchFixedFont' = ("Courier",8pt)
     'TitleFont2' = ("Courier",8pt)
     'TitleFont' = ("Courier",8pt)
     'StrongFont' = ("Courier",8pt)
     'EmphasisFont' = ("Courier",8pt)
     'FixedEmphasisFont' = ("Courier",8pt)
     'FixedStrongFont' = ("Courier",8pt)
     'FixedHeadingFont' = ("Courier",8pt)
     'FixedFont' = ("Courier",8pt)
     'headingEmphasisFont' = ("Courier",8pt)
     'headingFont' = ("Courier",8pt)
     'docFont' = ("Courier",8pt);
    style table from output /
```

```
    cellpadding = 2pt
    cellspacing = 0pt
    background=white
    rules=groups
    frame=void;
  style color_list from color_list /
   'link' = black
   'bgH' = white
   'fg' = black
   'bg' = white;
 end;
RUN;
```

Compare the output for RTF, PDF and HTML to the listing output. A few differences still exist, mostly between heading columns in HTML output versus the other destinations. But overall the ODS output has the look and feel of listing output. The output for the ODS destinations PDF, RTF and HTML are shown in Figures 16.21, 16.22, and 16.23 respectively. Output sent to the LISTING destination is shown in Figure 16.24.

Figure 16.21: PDF Output

Simulate Listing Look

Effect on Traffic	Event	Location	Highway
Major Jam	Injury	San Francisco	280
Major Jam	Injury	San Francisco	80
Major Jam	Disabled	South Bay	101
Major Jam	Disabled	San Francisco	101
Major Jam	Disabled	San Francisco	80

Figure 16.22: RTF Output

Simulate Listing Look

Effect on Traffic	Event	Location	Highway
Major Jam	Injury	San Francisco	280
Major Jam	Injury	San Francisco	80
Major Jam	Disabled	South Bay	101
Major Jam	Disabled	San Francisco	101
Major Jam	Disabled	San Francisco	80

Figure 16.23: HTML Output

Simulate Listing Look

Effect on Traffic	Event	Location	Highway
Major Jam	Injury	San Francisco	280
Major Jam	Injury	San Francisco	80
Major Jam	Disabled	South Bay	101
Major Jam	Disabled	San Francisco	101
Major Jam	Disabled	San Francisco	80

Figure 16.24: Listing Output

Simulate Listing Look

Effect on Traffic	Event	Location	Highway
Major Jam	Injury	San Francisco	280
Major Jam	Injury	San Francisco	80
Major Jam	Disabled	South Bay	101
Major Jam	Disabled	San Francisco	101
Major Jam	Disabled	San Francisco	80

Hiding Procedure Titles

In this final example on modifying output structure, the TEMPLATE procedure is not required. This example uses an ODS statement to achieve the desired effect.

If you're creating your results for people who don't know SAS, they might find the procedure titles confusing. After all, "The FREQ Procedure" is meaningful to a SAS programmer, but not exactly common terminology. It would be nice to be able to get rid of these labels. In the table of contents and table of pages for HTML output, the ODS PROCLABEL statement enables you to modify these labels. They can be changed to something meaningful, or removed entirely.

However, the ODS PROCLABEL statement does not affect the corresponding procedure labels in the body file of your output. If you want to get rid of the label "The FREQ Procedure" that's displayed above your output, you have to use a different ODS statement.

The new statement is ODS NOPROCTITLE, whose shorter alias is NOPTITLE.

```
ODS NOPTITLE;
```

To see how it works, we'll create a simple PDF file. Figure 16.25 shows the output before using the new option. The standard title "The FREQ Procedure" appears above the output table.

To get rid of this title, just add the NOPTITLE option to an ODS statement, as in the following code.

```
    ODS PDF FILE='NoProcedure.pdf';
ODS NOPTITLE;
title 'Construction Materials';
proc freq data=Buildings;
   tables material*type / norow nocol nopercent;
run;
ODS PDF CLOSE;
```

Figure 16.26 shows the same table with the new option in place. The procedure title has disappeared. Notice that the title "Construction Materials" that was created by the TITLE statement in the code has not been affected.

The ODS NOPTITLE option persists until you change it back with an ODS PTITLE statement.

The PTITLE option affects only the procedure titles in the body of the output. If you use this technique with the HTML destination and create a CONTENTS file, you will still get the title "The FREQ Procedure" in the table of contents. To completely remove the title, you need to use the NOPTITLE option in conjunction with the PROCLABEL statement (as shown in Chapter 3, "HTML Output") and a custom style template to remove the blank lines from the table of contents (as shown in the previous example).

Figure 16.25

Construction Materials

The FREQ Procedure

Frequency

Table of Material by Type					
Material(Primary construction material)	Type(Building type)				
	Office	Residential	Retail	Warehouse	Total
Brick	18	36	8	14	76
Concrete	42	34	15	24	115
Steel	36	0	2	9	47
Wood	39	48	4	13	104
Total	135	118	29	60	342

Figure 16.26

Construction Materials

Frequency

Table of Material by Type					
Material(Primary construction material)	Type(Building type)				
	Office	Residential	Retail	Warehouse	Total
Brick	18	36	8	14	76
Concrete	42	34	15	24	115
Steel	36	0	2	9	47
Wood	39	48	4	13	104
Total	135	118	29	60	342

Chapter 17
Modifying Output Colors and Images

ODS enables you to create color virtually anywhere you want it. If you'd like a purple frequency distribution and an orange regression model, you can do it. You can create Web pages using psychedelic color schemes. You can also create output that matches the colors in your corporate logo.

This chapter describes how colors work in ODS. It covers how and where to identify them, and shows how to create style definitions with custom color schemes.

In addition, because sometimes output is not generated for the Web or for a color printer, we look at a color scheme designed for black and white output.

This chapter uses the data set PAINTS to illustrate the concepts. This data set is available on the author pages for this book at http://support.sas.com/authors.

For a color version of the output from this chapter, see the author pages for this book at http://support.sas.com/authors.

Exploring the Style Elements That Control Color

The colors in an ODS style definition are organized much like the fonts. A style element creates a list of colors that are used elsewhere in the output. To see how colors work, run the following PROC TEMPLATE code and examine the style elements that control color selections. Figure 17.1 shows selected portions of the output.

```
    PROC TEMPLATE;
  SOURCE Styles.Default;
RUN;
```

The first style element shown is the color_list element. It contains a list of color codes, each of which is assigned to a name. The codes are RGB values given in hexadecimal. See "Identifying Colors" on page 444 for more information about these codes. The color names are just a convenience. Foreground colors all start with "fg" and background colors all start with "bg". The letters and numbers following these codes have no special meaning; they're just used to uniquely identify each color.

The second style element is called colors, and it is used to assign color names to more descriptive names that are used in various parts of the output. For example, in the color_list style element, there is a color called fgA, which has a specific color of cx002288. In the colors style element, this color is assigned to notefg, proctitlefg, systitlefg, confolderfg, contitlefg, and docfg. So the color_list style element creates a list of colors, and the colors style element assigns these colors to a list of names representing various parts of your output. Thus far, all we have are two lists. The key to making this work is the third style element, which is shown in Figure 17.2. It is an example of how colors are applied to output. In this case, the color defined by systitlefg is applied to the TitlesAndFooters style element as the foreground color, and the color defined by systitlebg is applied to the TitlesAndFooters style element as the background color. The net effect of all of this code is that the titles and footnotes in your output are colored royal blue (xx002288) and placed on a background of light gray (cxE0E0E0).

If you want to modify the color scheme for the titles and footnotes, there are three ways to do this. First, you could define a new style template with a different color_list style element. If you change fgA and bgA to different hexadecimal color codes, the new colors are then assigned to the color names in the colors style element. These names would then be assigned to the TitlesAndFooters style element. The problem with this approach is that the new colors would also apply everywhere else that fgA and bgA are referenced. If you want to create a consistent change in the color scheme across all parts of your output, modifying the color_list style element is the way to go.

The second way to modify the color scheme is to modify the colors style element. You can change the color that is used by the systitlefg or systitlebg attributes. This approach works better if you want to change the color assignment for a particular part of your output.

The third way to modify the color scheme is to change the color name that is given in the style element that controls the part of the output you want to change. The TitlesAndFooters

style element can be modified to request a color name other than systitlefg or systitlebg.

You can also use a combination of the above. For example, if you wanted to create purple titles and footnotes, you could add a new color to the color_list style element, create a new color name for the new color in the colors style element, and assign the new color name to the appropriate attribute in the TitlesAndFooters style element. The examples in this chapter cover these techniques.

Figure 17.1

```
class color_list
   "Colors used in the default style" /
   'fgB2' = cx0066AA
   'fgB1' = cx004488
   'fgA4' = cxAAFFAA
   'bgA4' = cx880000
   'bgA3' = cxD3D3D3
   'fgA2' = cx0033AA
   'bgA2' = cxB0B0B0
   'fgA1' = cx000000
   'bgA1' = cxF0F0F0
   'fgA' = cx002288
   'bgA' = cxE0E0E0;
class colors
   "Abstract colors used in the default style" /
   'headerfgemph' = color_list('fgA2')
   'headerbgemph' = color_list('bgA2')
   'headerfgstrong' = color_list('fgA2')
   'headerbgstrong' = color_list('bgA2')
   'headerfg' = color_list('fgA2')
   'headerbg' = color_list('bgA2')
   'datafgemph' = color_list('fgA1')
   'databgemph' = color_list('bgA3')
   'datafgstrong' = color_list('fgA1')
   'databgstrong' = color_list('bgA3')
   'datafg' = color_list('fgA1')
   'databg' = color_list('bgA3')
   'batchfg' = color_list('fgA1')
   'batchbg' = color_list('bgA3')
   'tableborder' = color_list('fgA1')
   'tablebg' = color_list('bgA1')
   'notefg' = color_list('fgA')
   'notebg' = color_list('bgA')
   'bylinefg' = color_list('fgA2')
   'bylinebg' = color_list('bgA2')
   'captionfg' = color_list('fgA1')
   'captionbg' = color_list('bgA')
   'proctitlefg' = color_list('fgA')
   'proctitlebg' = color_list('bgA')
   'titlefg' = color_list('fgA')
   'titlebg' = color_list('bgA')
   'systitlefg' = color_list('fgA')
   'systitlebg' = color_list('bgA')
   'Conentryfg' = color_list('fgA2')
   'Confolderfg' = color_list('fgA')
   'Contitlefg' = color_list('fgA')
   'link2' = color_list('fgB2')
   'link1' = color_list('fgB1')
   'contentfg' = color_list('fgA2')
   'contentbg' = color_list('bgA2')
   'docfg' = color_list('fgA')
   'docbg' = color_list('bgA');
```

Figure 17.2

```
class TitlesAndFooters /
   font = Fonts('TitleFont2')
   backgroundcolor = colors('systitlebg')
   color = colors('systitlefg');
```

Modifying the Color of Your Titles

Because there are so many ways to modify color settings, it can get a bit confusing. This first example is very simple so you can see what is going on.

The default template Styles.Default uses dark text on a light background for titles and footnotes. For a different look, we can reverse those settings. In Styles.Default, fgA is set to the color cx002288, and bgA is set to the color cxE0E0E0. To reverse the settings, you just have to exchange those two values. The following code shows how to define a new template where the color_list style element now has these new settings:

```
PROC TEMPLATE;
DEFINE STYLE ReverseColors;
PARENT=Styles.Default;
CLASS color_list /
   'fgB2' = cx0066AA
   'fgB1' = cx004488
   'fgA4' = cxAAFFAA
   'bgA4' = cx880000
   'bgA3' = cxD3D3D3
   'fgA2' = cx0033AA
   'bgA2' = cxB0B0B0
   'fgA1' = cx000000
   'bgA1' = cxF0F0F0
   'fgA' = cxE0E0E0
   'bgA' = cx002288;
END;
RUN;
```

Figure 17.3 shows a simple table created with Styles.Default. Figure 17.4 shows the same table created with our new ReverseColors template. Notice how the entire background has been reversed. This is because fgA and bgA apply to more than just the titles and footnotes.

Another way to write this code so that only the titles and footnotes are affected is to just reverse the color assignments in the TitlesAndFooters style element, as shown in the following code:

```
PROC TEMPLATE;
DEFINE STYLE RevColorTitle;
PARENT=Styles.Default;
CLASS TitlesAndFooters /
   font = Fonts('TitleFont2')
   background = colors('systitlefg')
   foreground = colors('systitlebg');
END;
RUN;
```

This code assigns the background color to the foreground and the foreground color to the background. The result is output with the title shown in reverse, as shown in Figure 17.5.

Figure 17.3

Paint Industry Statistics

Frequency

Table of Base by Type				
	Type(Type of Finish)			
Base(Base)	Paint	Sealer	Stain	Total
Latex	256	95	157	508
Oil	115	46	98	259
Total	371	141	255	767

Figure 17.4

Paint Industry Statistics

Frequency

Table of Base by Type				
	Type(Type of Finish)			
Base(Base)	Paint	Sealer	Stain	Total
Latex	256	95	157	508
Oil	115	46	98	259
Total	371	141	255	767

Figure 17.5

Paint Industry Statistics

Frequency

Table of Base by Type				
	Type(Type of Finish)			
Base(Base)	Paint	Sealer	Stain	Total
Latex	256	95	157	508
Oil	115	46	98	259
Total	371	141	255	767

Identifying Colors

The default style template Styles.Default uses colors such as cx0066AA, cx004488, and cx002288. What do these stand for? The "cx" part indicates that this is an RGB color given in hexadecimal values. The numbers that follow are the red, green, and blue values that define the colors. For example, cx0066AA refers to red=00, green=66, and blue=AA. Those values produce a medium grayish blue. In this color scheme, cxFFFFFF refers to white, cx000000 refers to black, and cx0000FF refers to blue.

If you want to change a color on the color list, you can simply replace the old RGB color code with the new RGB color code. For example, if you change the code for the fgA value, you can request green foreground text. Here is the code change:

```
"fgA" = cx002288      to      "fgA" = cx00FF00
```

This creates the output shown in Figure 17.6. If you look at the generated HTML code in detail, you will see the following color assignment for the title:

```
font-family: Arial, Helvetica, sans-serif; font-size: large;
font-weight: bold; font-style: italic; color: #00FF00;
```

The color code is passed through to the HTML code for your Web page. Instead of cx00FF00, it is listed in the HTML format of #00FF00, but the code is the same. If you create RTF or Printer output, the same color is used, though you can't see the color code directly like this.

If you find these color codes confusing, you can also choose to identify your colors by name. This can make your style definition code more readable. For example, to get green foreground text, we can use the following code:

```
"fgA" = GREEN
```

Figure 17.7 shows the results with the new color. This example illustrates the limitations of color names. The green color selected by GREEN is not the same as the green color selected by cx00FF00. If you look at the HTML code, you can see why this is the case.

```
font-family: Arial, Helvetica, sans-serif; font-size: large;
font-weight: bold; font-style: italic; color: #008000;
```

SAS has assigned the color code cx008000 instead of cx00FF00 when creating the HTML output. This is the catch to using color names. SAS assigns the names to color values when it generates the output, and you can only create colors for which SAS has an assigned name. For a list of the color names and their values, run the following code:

```
    proc registry list
  startat="COLORNAMES";
 run;
```

If you're a stickler for getting the exact color you want, you might be happier using the RGB codes. You can download a tool from the Internet to help you pick colors. Just use any search engine to search on the term "Color Picker." These tools generally have a color palette you can click on to get the RGB code for that color. Figure 17.8 shows an example of one of

these tools. By clicking on the box with the desired shade of green, you can look up the code 129118 that will give you that shade of green.

Figure 17.6

Paint Industry Statistics

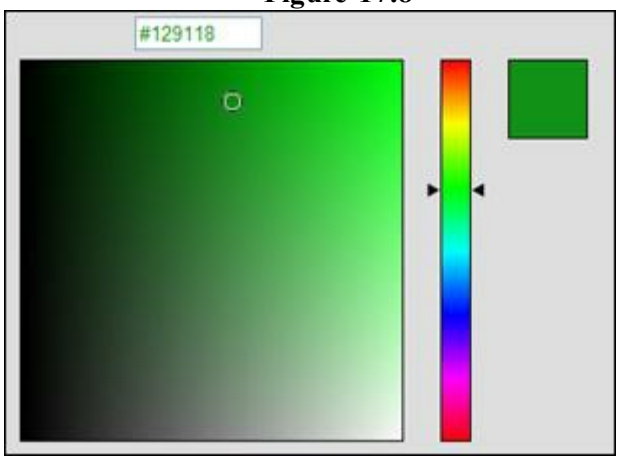
Analysis Variable : Warranty Year of Warranty				
N	Mean	Std Dev	Minimum	Maximum
767	6.4	2.8	2.0	11.0

Figure 17.7

Paint Industry Statistics

Analysis Variable : Warranty Year of Warranty				
N	Mean	Std Dev	Minimum	Maximum
767	6.4	2.8	2.0	11.0

Figure 17.8



Most color pickers show you the 216 "safe" colors to use for HTML. These colors are generally rendered correctly by all browsers. Other colors might look different depending on your choice of browser.

See the author pages for this book for a link to sample code that builds an interactive display of all of the SAS named colors.

Modifying Background Colors

Each of the previous examples in this chapter has looked at changing the color of the text in your output. This example focuses on the background.

If you expect that users will want to print your HTML output a lot, you might want to modify the background color in your style template. That gray background looks fine when viewed in a browser, but it doesn't look as good when you print. Because your printer cannot print all of the way to the edge of the paper, you get output on a gray background with a white border.

You can change the background color in the color_list style element to use a white background. In the following code, the bgA color is changed from cxE0E0E0, a gray shade, to cxFFFFFF, a pure white.

```
    PROC TEMPLATE;
  DEFINE STYLE WhiteBack;
  PARENT=Styles.Default;
  CLASS color_list /
     'fgB2' = cx0066AA
     'fgB1' = cx004488
     'fgA4' = cxAAFFAA
     'bgA4' = cx880000
     'bgA3' = cxD3D3D3
     'fgA2' = cx0033AA
     'bgA2' = cxB0B0B0
     'fgA1' = cx000000
     'bgA1' = cxF0F0F0
     'fgA' = cx002288
     'bgA' = cxFFFFFF;
   END;
 RUN;
```

This style definition changes your output from the color scheme shown in Figure 17.9 to the color scheme shown in Figure 17.10. Now the entire background, except the tables themselves, is shown in white.

If you want the background to be completely white, including the table, then you need to make a few more changes. Every color that starts with "bg" needs to be changed to cxFFFFFF. Here are the items to change:

```
     'bgA4' = cxFFFFFF
     'bgA3' = cxFFFFFF
     'bgA2' = cxFFFFFF
     'bgA1' = cxFFFFFF
     'bgA' = cxFFFFFF
```

The revised output is shown in Figure 17.11. Now everything is white except the foreground text and the table borders.

Figure 17.9

Paint Industry Statistics

Analysis Variable : Warranty Year of Warranty						
Type of Finish	N Obs	N	Mean	Std Dev	Minimum	Maximum
Paint	371	371	6.5	2.8	2.0	11.0
Sealer	141	141	6.4	3.0	2.0	11.0
Stain	255	255	6.4	2.8	2.0	11.0

Figure 17.10

Paint Industry Statistics

Analysis Variable : Warranty Year of Warranty						
Type of Finish	N Obs	N	Mean	Std Dev	Minimum	Maximum
Paint	371	371	6.5	2.8	2.0	11.0
Sealer	141	141	6.4	3.0	2.0	11.0
Stain	255	255	6.4	2.8	2.0	11.0

Figure 17.11

Paint Industry Statistics

Analysis Variable : Warranty Year of Warranty						
Type of Finish	N Obs	N	Mean	Std Dev	Minimum	Maximum
Paint	371	371	6.5	2.8	2.0	11.0
Sealer	141	141	6.4	3.0	2.0	11.0
Stain	255	255	6.4	2.8	2.0	11.0

Modifying Colors in Tables

If you want to change all of the table elements to the same color scheme, then modifying table colors is simple. As in the previous example, you can just change all of the colors that start with “bg” to the same color. You could also do the same thing with all of the colors labeled “fg”.

However, what makes tables interesting is the mix of colors used to define the table cells and headings. Let’s assume that we’re creating a table for a company with a green and gold logo. This example creates a color scheme to match.

In order to change the colors used by the table, we first need to find out which colors are used in the table. The way to do this is run PROC TEMPLATE with a SOURCE statement for the default template you want to adapt. This enables you to start tracking down each of the parts of the table. Starting with the header, if you look at the HeadersAndFooters style element, you will find that the colors used are headerfg for the foreground text, and headerbg for the background. If you then look these up in the color_list style element, you find that headerfg is assigned color fgA2 and headerbg is assigned color bgA2.

Following the same steps, you will learn that the table cells use fgA1 for the foreground and bgA3 for the background. The table border color is fgA and the table background (the color that shows through when CELLSPACING= is set to a value greater than 0) is bgA1.

To have a new color scheme in your new style template, all you have to do is change those five color settings to colors that match the green and gold logo. When replacing the colors, the main thing to keep in mind is to replace dark colors with dark colors, and light colors with light colors. Otherwise, you might get output with text that is hard to distinguish from the background. The revised colors in the following SAS 9.2 code create a table color scheme with green text on a gold background. The table text and borders are a shade of dark green.

```
    PROC TEMPLATE;
  DEFINE STYLE DefaultLogoColors;
  PARENT=Styles.Default;
  CLASS color_list /
    'fgB2' = cx0066AA
    'fgB1' = cx004488
    'fgA4' = cxAAFFAA
    'bgA4' = cx880000
    'bgA3' = cxFFFF33
    'fgA2' = cx336600
    'bgA2' = cxFFCC00
    'fgA1' = cx003300
    'bgA1' = cxFFFF66
    'fgA' = cx002288
    'bgA' = cxE0E0E0;
  END;
RUN;
```

Compare Figure 17.12 to Figure 17.13 to see the change. Taking this color scheme a step

further, you could also modify fgA and bgA to create titles and a page background in a similar color scheme, as shown in Figure 17.14. The colors used were cx336600 for the foreground and cxFFFF66 for the background.

Figure 17.12

Paint Industry Statistics

Analysis Variable : Warranty Year of Warranty						
Type of Finish	N Obs	N	Mean	Std Dev	Minimum	Maximum
Paint	371	371	6.5	2.8	2.0	11.0
Sealer	141	141	6.4	3.0	2.0	11.0
Stain	255	255	6.4	2.8	2.0	11.0

Figure 17.13

Paint Industry Statistics

Analysis Variable : Warranty Year of Warranty						
Type of Finish	N Obs	N	Mean	Std Dev	Minimum	Maximum
Paint	371	371	6.5	2.8	2.0	11.0
Sealer	141	141	6.4	3.0	2.0	11.0
Stain	255	255	6.4	2.8	2.0	11.0

Figure 17.14

Paint Industry Statistics

Analysis Variable : Warranty Year of Warranty						
Type of Finish	N Obs	N	Mean	Std Dev	Minimum	Maximum
Paint	371	371	6.5	2.8	2.0	11.0
Sealer	141	141	6.4	3.0	2.0	11.0
Stain	255	255	6.4	2.8	2.0	11.0

Adding a New Color to the Color Scheme

If you would like to depart from the default color scheme and add a new color for some part of the output, you can do that by adding a new color to your style definition.

The following code creates a new purple color just for footnotes (the code has been abbreviated to save space, but the important changes are shown). First, the new color is added to the "color_list" style element. This new color is called fgNew and its color code is set to cx660066, which is a dark purple. Second, the new color is assigned to the name footnotefg in the colors style element. Finally, the new color name is assigned to the FOREGROUND= attribute in the SystemFooter style element. Previously, this style element contained only the FONT= attribute. The foreground setting has been added so we can modify the color.

```
     PROC TEMPLATE;
  DEFINE STYLE DefaultPurpleFN;
  PARENT=Styles.Default;
  CLASS color_list /
          ... list standard colors here ...
     'fgNew'= cx660066;
  CLASS colors /
          ... list standard colors here ...
     'footnotefg' = color_list('fgNew');
  CLASS SystemFooter /
     font = Fonts('TitleFont')
     foreground = colors('footnotefg');
   END;
 RUN;
```

Using the style template DefaultPurpleFN, you can change your output from the version shown in Figure 17.15 to the version shown in Figure 17.16. The only difference is that the second version has a purple footnote.

There is a simpler way to create the same effect. Instead of modifying the color_list and colors style elements, the new color could be assigned directly in the SystemFooter style element. All you have to do is list the color code as the foreground setting, as in the code below.

```
     PROC TEMPLATE;
  DEFINE STYLE DefaultPurpleFN;
  PARENT=Styles.Default;
  CLASS SystemFooter /
     font = Fonts('TitleFont')
     foreground = cx660066;
   END;
 RUN;
```

This code produces exactly the same output as the previous code. However, it makes your source code more confusing because it does not follow the convention of assigning colors in the color_list and colors style elements. You can decide which approach you prefer.

Figure 17.15

Paint Industry Statistics

Variable: Price

Basic Statistical Measures			
Location		Variability	
Mean	11.07898	Std Deviation	2.86897
Median	11.01149	Variance	8.23099
Mode	.	Range	9.97714
		Interquartile Range	5.00348

Current Month Sales

Figure 17.16

Paint Industry Statistics

Variable: Price

Basic Statistical Measures			
Location		Variability	
Mean	11.07898	Std Deviation	2.86897
Median	11.01149	Variance	8.23099
Mode	.	Range	9.97714
		Interquartile Range	5.00348

Current Month Sales

Designing a Template to Enhance Accessibility

Some color schemes look great in color, but when they are converted to gray scale, there is not enough contrast between the foreground and background and the results are hard to read. When preparing output for visually impaired viewers, accessibility experts consider black and white, high-contrast output to be the best choice. Just as SAS comes with a multitude of color-coordinated style templates, it also comes with several black and white style templates. So, if you are designing a template for accessibility purposes, it is probably better to start from one of these style templates.

Three black and white styles are HighContrast, Journal, and MonochromePrinter. The Pearson Corr tables in Figures 17.17, 17.18, and 17.19 show an example of each style.

Another accessibility consideration is whether your report will be viewed on the screen or printed on paper. The chances are that both will happen. The general rule of thumb for font usage is that sans-serif fonts, such as Arial or Helvetica, are best for screen viewing, whereas serif family fonts, such as Times New Roman or Courier, are best for printed output. As you can see in the figures, Journal and HighContrast both use sans-serif fonts, and MonochromePrinter uses a serif font.

Let's say that you like the look of the table with the HighContrast style and you plan to use that style template as is for screen viewing. But, you want to create an equivalent custom style that uses a serif font for printing. Your template code would look like this:

```
PROC TEMPLATE;
  DEFINE STYLE HCSerif;
  PARENT=Styles.HighContrast;
  CLASS fonts /
    'TitleFont2' = ("<MTserif>, Times Roman",4,bold italic)
    'TitleFont' = ("<MTserif>, Times Roman",5,bold italic)
    'StrongFont' = ("<MTserif>, Times Roman",4,bold)
    'EmphasisFont' = ("<MTserif>, Times Roman",3,italic)
    'FixedEmphasisFont' = ("<MTmonospace>, Courier",2,italic)
    'FixedStrongFont' = ("<MTmonospace>, Courier",2,bold)
    'FixedHeadingFont' = ("<MTmonospace>, Courier",2)
    'BatchFixedFont' = ("SAS Monospace, <MTmonospace>, Courier",2)
    'FixedFont' = ("<MTmonospace>, Courier",2)
    'headingEmphasisFont' = ("<MTserif>, Times Roman",4,bold italic)
    'headingFont' = ("<MTserif>, Times Roman",4,bold)
    'docFont' = ("<MTserif>, Times Roman",3);
   END;
RUN;
```

Because the HighContrast style template inherits all of its attributes from Styles.Default, the font section from Styles.Default was copied and modified to use the MTserif SAS registry font or Times Roman for the main font attributes. Notice that the font size was left as a relative number, although an absolute point size could also have been used. HTML output from this template is shown in Figure 17.20. Alternately, you could modify one of the high contrast serif-based style templates to use sans-serif fonts. Although you could design your own black and white style template from scratch, working from an existing style template will

save you some time and effort.

Figure 17.17: HTML HighContrast Style

Pearson Correlation Coefficients, N = 767 Prob > |r| under H0: Rho=0	
	Price
Warranty Year of Warranty	0.01597 0.6588

Figure 17.18: RTF Journal Style

Pearson Correlation Coefficients, N = 767 *Prob > |r| under H0: Rho=0*	
	Price
Warranty Year of Warranty	0.01597 0.6588

Figure 17.19: PDF MonochromePrinter Style

Pearson Correlation Coefficients, N = 767 Prob > |r| under H0: Rho=0	
	Price
Warranty Year of Warranty	0.01597 0.6588

Figure 17.20: Custom HCSerif Style (ODS HTML)

Pearson Correlation Coefficients, N = 767 Prob > |r| under H0: Rho=0	
	Price
Warranty Year of Warranty	0.01597 0.6588

In addition to these recommendations, there are specific recommendations for meeting Rehabilitation Act Section 508 accessibility requirements. In earlier versions of SAS, you needed to use a custom markup tagset template in order to create 508-compliant HTML pages. In SAS 9.2, all of those accessibility enhancements have been added to the HTML destination. SAS documents additional information about Section 508 accessibility requirements on their support Web site, support.sas.com.

Adding a Logo to Your Output

Once you start customizing your style template, a common thing to want to do is add a company logo to your page headings. This is a great way to add your corporate identity to any results you send out. This example shows how to add a logo, and then demonstrates the technique for HTML, RTF, and Printer output.

The first step is to define a new style template based on an existing default template and identify the style element and its attribute that can insert an image into the output. In Styles.Default, the style element that can insert an image is Body, and the attribute to name an image file to be used is PREIMAGE=. For HTML output, these images must be in a browser-compatible format such as GIF, PNG, or JPEG. For RTF output, the images need to be PNG or JPEG files. For Printer output (PostScript, PDF), you can use virtually any image format. The following code defines the new template CorpLogo.

```
    PROC TEMPLATE;
  DEFINE STYLE CorpLogo;
  PARENT=Styles.Default;
  CLASS Body /
     preimage="corporatelogo.gif";
   END;
 RUN;
```

Figure 17.21 shows the results from the new style definition in HTML. The logo is displayed at the top of the page. This example is just the beginning of the process of creating a corporate style template. The red logo doesn't fit in well with the blue and gray default style. A bit more tweaking of the style, picking a more appropriate parent style (e.g., Journal), and changing the title color to match the logo yields a better result. The following code produces Figure 17.22.

```
    PROC TEMPLATE;
  DEFINE STYLE CorpLogoJournal;
  PARENT=Styles.Journal;
  CLASS Body /
     preimage="corporatelogo.gif";
  CLASS SystemTitle /
     foreground = cx990000;
  END;
 RUN;
```

It's a bit of work to create a custom corporate style, but after it is created, you can use this style repeatedly to ensure a consistent corporate look for all of your output.

Figure 17.21

October Price Report

Analysis Variable : Price				
N	Mean	Std Dev	Minimum	Maximum
767	11.08	2.87	6.01	15.99

Figure 17.22

October Price Report

Analysis Variable : Price				
N	Mean	Std Dev	Minimum	Maximum
767	11.08	2.87	6.01	15.99

Adding a Logo and a Custom Heading to Your HTML Output

Most style element definition techniques work for all output formats, but if you are creating HTML output, there are a few extra tricks in your bag. You can add images and HTML code to your HTML output.

As an example, we will set up a logo and company name as a default page heading. This time, instead of the PREIMAGE= attribute, we'll use the PREHTML= attribute of the Body style element. The attribute enables you to pass HTML code straight through to your output. In this case, ODS passes to HTML three kinds of items: the IMG= tag used to display an image; some text and formatting tags to create a heading; and some table tags to organize the logo and text. The BORDER="0" setting hides the border of the image so that it blends into the Web page. The SRC= "logo.gif" names the image to be used. These images must be in a browser-compatible format such as GIF, PNG, or JPEG. The following code defines new style template CORPLOGO2:

```
    PROC TEMPLATE;
  DEFINE STYLE CorpLogo2;
  PARENT=Styles.Default;
  CLASS Body /
    prehtml='<table width=100%><tr><td nowrap align=left>
      <img border="0" src="corporatelogo.gif">
      <font face="Arial" size=3><b>SprayTech Corp.</b></font>
      </tr></td></table>';
  END;
 RUN;
```

The results are shown in Figure 17.23. Now both the logo and the company name appear at the top of the page.

Using the PREHTML= attribute, you can add anything you want to the top of your output. However, this technique only works for the HTML destination.

There is also a matching POSTHTML= attribute that enables you to add custom HTML code to the bottom of your output. This is a handy way to put a standard footer on all of your output pages.

Figure 17.23

SprayTech Corp.

October Price Report

Analysis Variable : Price				
N	Mean	Std Dev	Minimum	Maximum
767	11.08	2.87	6.01	15.99

Adding a Background Image to Your HTML Output

The previous example showed how to add a logo at the top of each output page. This example shows how to use an image for the background. We will define a style template with a repeated image of the company logo as a background. Again, we will be working with the Body style element. This time, instead of the PREHTML= attribute, we will take advantage of another attribute, BACKGROUNDIMAGE=. This attribute enables you to specify an image file to use as the page background. You do not have to write the HTML code yourself if all you want is a background image. The following code defines the new style template CorpLogoBG.

```
     PROC TEMPLATE;
  DEFINE STYLE CorpLogoBG;
  PARENT=Styles.Default;
  CLASS Body /
     backgroundimage="corporatelogobg.gif";
  END;
 RUN;
```

The resulting Web output is shown in Figure 17.24. The output is very ugly because the image clashes with the Styles.Default color scheme, but you can get the idea. We can enhance this output by adding the company title to the page heading and by fixing the ugly background behind the page title. The company title is added in exactly the same way as in the previous example. The PREHTML= attribute is used to add the HTML needed to create the title. Next, we can fix the background for the titles. This requires removing the background setting from the TitlesAndFooters and Container style elements. The following code defines the new style template CorpLogoBG2.

```
     PROC TEMPLATE;
  DEFINE STYLE CorpLogoBG2;
  PARENT=Styles.Default;
  CLASS Body /
     prehtml='<table width=100%><tr><td nowrap align=left>
       <font face="Arial" size=5><b>SprayTech Corp.</b></font>
       </tr></td></table>'
     backgroundimage='corporatelogobg.gif';
   CLASS Container /
     font = Fonts('DocFont')
     foreground = colors('docfg');
   CLASS TitlesAndFooters /
     font = Fonts('TitleFont2')
     foreground = colors('systitlefg');
   END;
 RUN;
```

Now we get the table shown in Figure 17.25. This looks better, but it is still ugly. One solution to this problem would be to use the Styles.Minimal template as the parent, rather than Styles.Default. This requires an additional code tweak to change the table background to white, but can yield results like Figure 17.26. The bottom line is that no matter how you create your background image style definition, you’re going to have to tweak some settings

(color scheme, background colors) to get the attractive look you want. This example is merely an introduction.

Figure 17.24

October Price Report

Analysis Variable : Price				
N	Mean	Std Dev	Minimum	Maximum
767	11.08	2.87	6.01	15.99

Figure 17.25

SprayTech Corp.

October Price Report

Analysis Variable : Price				
N	Mean	Std Dev	Minimum	Maximum
767	11.08	2.87	6.01	15.99

Figure 17.26

SprayTech Corp.

October Price Report

Analysis Variable : Price				
N	Mean	Std Dev	Minimum	Maximum
767	11.08	2.87	6.01	15.99

Adding Images to RTF Output (with Templates)

One of the quickest and easiest ways to dress up your RTF reports is to drop in a logo or graphic on each page. All you need to do is find a JPEG or PNG or other type of graphic to use, and write a simple PROC TEMPLATE step to create a style using your graphic. In an earlier chapter, we illustrated the use of logo insertion using ODS ESCAPECHAR.

Another way to insert a logo at the top of the first page of your output is to define a new ODS style and then reference the style in the first ODS RTF statement of your program. The following code defines the style RTFImage, which copies all the attributes from the RTF style and modifies the PREIMAGE= attribute of the Body style element. The attribute PREIMAGE= specifies the filename of the logo graphic. The filename specification must include path information that points to the location of the graphic file when the program is run.

```
    PROC TEMPLATE;
  DEFINE STYLE RTFImage;
    PARENT=Styles.RTF;
    CLASS Body /
      preimage="corporatelogo.jpg";
  END;
RUN;
```

The program that follows references the style RTFImage in the ODS RTF statement and is submitted after the PROC TEMPLATE step above. The resulting output is shown in Figure 17.27. ODS centers the graphic at the top of the first page. Subsequent pages do not show the logo. Because ODS embeds the image in the RTF file, you do not need to worry about keeping the graphics file along with the output file as you would if you had sent the output to the HTML destination. The ODS NOPROCTITLE statement suppresses the procedure title.

```
    ODS NOPROCTITLE;
ODS RTF FILE='CorpLogoTemp.rtf' STYLE=RTFImage;
proc means data=paints n mean std min max maxdec=2;
      title "October Price Report";
      var price;
    run;

ODS RTF CLOSE;
```

Assume that the SAS system option CENTER is in effect. This setting centers both the output and the graphic. You can allow SAS to place your graphic based on the SAS system option setting, or you can specify a justification setting in your style. The following code shows how to revise the Body style element so that ODS left-justifies the graphic, regardless of the setting of the CENTER system option. The justification attribute JUST= specifies a value of LEFT. The revised output is shown in Figure 17.28.

```
    CLASS Body /
  just=left
  preimage="corporatelogo.jpg";
```

Figure 17.27

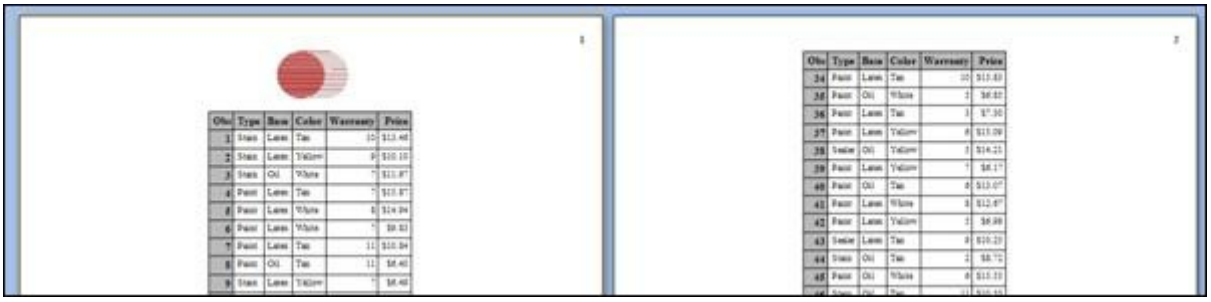
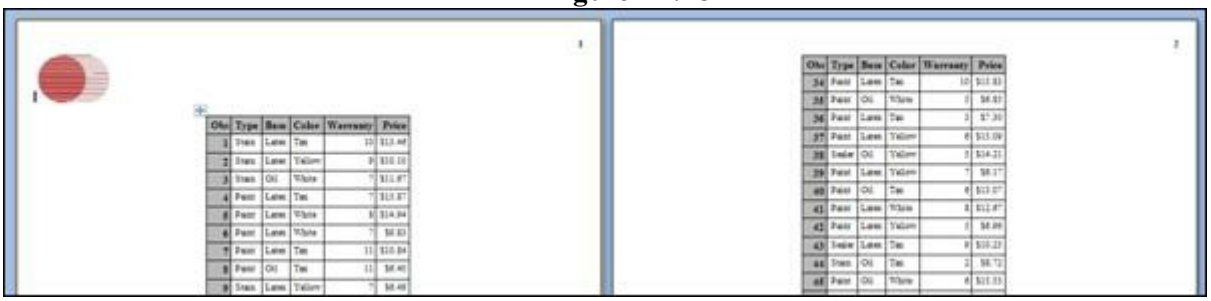


Figure 17.28



Adding Images to Each Page of Your RTF Output

The previous programs inserted a single graphic at the top of the first page. If you'd like to repeat the graphic on every page, all you need is two modifications. First, you need to modify a different style element and delete the modification for the Body style element:

```
    CLASS SystemTitle  /
just=left
preimage="corporatelogo.jpg";
```

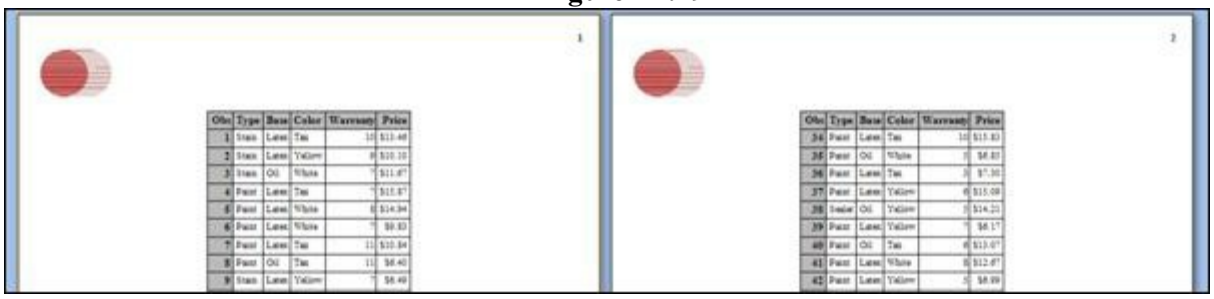
The second change you need to make is to include at least one TITLE statement in your program, even if it only creates a blank title:

```
TITLE ' ';
```

The output produced by this last version of the RTFImage style template is shown in Figure 17.29. The graphic now appears in every page. You can also use this technique to put a graphic in every page footer by using the SystemFooter style element and including at least one FOOTNOTE statement.

A downside of modifying the SystemTitle style element as shown above is that the image is inserted before the text for each TITLE statement.

Figure 17.29



Adding Images to Printer Output (with Templates)

This technique is basically the same as the example shown for RTF output, except that you have a wider choice of graphics file types. The PROC TEMPLATE step is the same as in the RTF example. Just change the PARENT setting from the RTF style to the PRINTER style.

```
    PROC TEMPLATE;
  DEFINE STYLE PrinterLogo;
  PARENT=Styles.Printer;
  CLASS body  /
    preimage='corporatelogo.jpg';
  END;
RUN;
```

If you generate Printer output using the PrinterLogo style, you will see the other difference between RTF and Printer output. The style defined above creates the results shown in Figure 17.30. Here is the program:

```
ODS NOPROCTITLE;
ODS PRINTER STYLE=PrinterLogo;
proc means data=paints n mean std min max maxdec=2;
  title "October Price Report";
  var price;
run;
ODS PRINTER CLOSE;
```

The graphic image is placed at the top of the first page, but instead of the output following directly beneath, ODS inserts a page break. To prevent this from happening, you need to turn off page breaks using the STARTPAGE= option, as shown below. This revised ODS PRINTER statement creates the output shown in Figure 17.31.

```
ODS PRINTER STYLE=PrinterLogo STARTPAGE=NO;
```

This option works fine as long as your output is only one page long, or if you don't care about the remaining procedures being combined on the same page. With STARTPAGE=NO in effect, SAS will not insert a page break until it runs out of room.

Another problem with this output is that the horizontal justification (left, center) for the logo is based on the SAS system option CENTER|NOCENTER. With CENTER=ON, the title text is overwritten with the logo. Unlike RTF output, this can't be fixed by setting JUST=L in the Body style element in the template. To fix this, toggle the option setting before and after the ODS PRINTER statement.

```
    options nocenter;
ODS PRINTER STYLE=PrinterLogo STARTPAGE=NO;
options center;
* the procedure(s) go here;
```

Figure 17.30

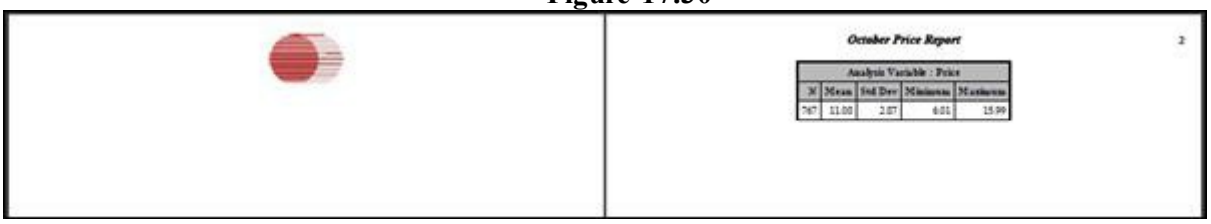


Figure 17.31

1

Analysis Variable : Price				
N	Mean	Std Dev	Minimum	Maximum
767	11.08	2.87	6.01	15.99

Figure 17.32

1

October Price Report

Analysis Variable : Price				
N	Mean	Std Dev	Minimum	Maximum
767	11.08	2.87	6.01	15.99

Another solution to both the page-break and logo justification problems is to add the logo via an escape character style override in a TITLE statement. This way you don't need to create and specify a new style, you don't need to include the STARTPAGE=NO option, and you don't need to change the CENTER=ON SAS system option. This code creates the output shown in Figure 17.32. Notice that with CENTER=ON, the title text "October Price Report" is centered over the table, while the logo is right-justified.

```
ODS ESCAPECHAR='^';
ODS PRINTER;
title "October Price Report" justify=right
"^{style[preimage='corporatelogo.jpg']}";
```

Chapter 18
Doing More with Styles and ODS

There is more that you can do with ODS and styles. You might want to define a custom style template and link it to a custom table definition for specific report purposes. Or you might need to design a custom style template for conformance with regulatory agency requirements. You might have a corporate standard to use only CSS style sheets for output files.

In some instances, you would design a custom style template using style template inheritance. Before SAS 9.2, it was harder to build an original style template, because you had to trace style element inheritance to determine the style elements that were going to be called by your procedure.

Beginning in SAS 9.2, there is an implicit parent style template in the SASHELP.TMPLMST item store called Base.Template.Style, which contains the inheritance model used for all style templates, even your custom templates and SAS built-in style templates. In this way, the SAS developers guarantee that a parent style element is available for your use that contains the style elements used by any procedure. In addition, if future SAS processes and procedures require new abstract style elements, SAS can add them to Base.Template.Style without affecting existing style templates.

Another feature that can help you with building custom style definitions is the ability to import Cascading Style Sheet (CSS) files into ODS style template format. Also, the new CSSSTYLE= option in the ODS statement for the RTF, PDF, and HTML destinations enables you to reference a CSS file directly instead of using a style template definition.

This chapter uses the data sets BILLINGS, CLINICAL, GALLERY, and HR to illustrate the concepts. These data sets are available on the author pages for this book at http://support.sas.com/authors.

Using the CSSSTYLE= Option

Corporate style guidelines are not just for Web pages, but can also apply to other types of documents, such as PDF files, Word documents, PowerPoint presentations, brochures, direct mailing letters, and so on. If your company has a standard CSS file available, you can apply it to your ODS output.

In the past, you generally needed one style template for HTML pages and a different style template for PDF and RTF files. You could have used your corporate CSS file for HTML output (using the STYLESHEET= option), but not for ODS PDF or RTF output.

Starting with SAS 9.2, however, you can use a CSS file directly with the RTF and PDF destinations, as well as with the HTML destination. So in this section, we show you how to reference a CSS file using an ODS option. Then, in an upcoming section, we show you how to

import a CSS file into a SAS style template. The CSSSTYLE= option has similar syntax to the STYLESHEET= option:

```
ODS HTML FILE='Out_18_1.HTML' CSSSTYLE='corp.css';
ODS PDF  FILE='Out_18_2.PDF'  CSSSTYLE='corp.css';
ODS RTF  FILE='Out_18_3.RTF'  CSSSTYLE='corp.css';
title 'Using Corporate CSS file';
proc print data=billings(obs=10);
  var CustomerName JobType Hours;
run;
ODS _ALL_ CLOSE;
```

Figure 18.1 shows the HTML results. The PDF and RTF results are shown in Figure 18.2 and 18.3 respectively. The complete text of the CORP.CSS file is shown below:

```
.Body{margin-left: 8px;
     margin-right: 8px;}
Table,.Table{font-family: Times New Roman, Times Roman, Courier New;
   border-collapse: collapse;
   border-color: #ff00FF #ff00FF #ff00FF #ff00FF;
   border-style: solid; border-width: 1px;}
th, .Header, .RowHeader{font-family: Times New Roman, Times Roman, Courier New;
   border-color: #0000ff #0000ff #0000ff #0000ff;
   padding: 2px; background: black; color: white;
   border-style: solid;  border-width: 1px;}
td, .Data{font-family: Times New Roman, Times Roman, Courier New;
   border-color: #000000 #000000 #000000 #000000;
   padding: 2px; border-style: solid;
   border-width: 1px;}
.D1{border-color: #000000;
   padding: 2px; background: #ffffff;
```

```
    border-style: solid; border-width: 1px;}
.D2{border-color: #000000;
    padding: 2px; background: #dddddd;
    border-style: solid; border-width: 1px;}
```

For more information about CSS style properties and how they work, consult any CSS guide, the W3C Web site at http://www.w3.org/Style/CSS/, or a CSS tutorial.

Figure 18.1

Using Corporate CSS file

Obs	CustomerName	JobType	Hours
1	Smith & Smith	Consulting	8.0
2	Sweat Shops Athletic	Consulting	4.0
3	Sweat Shops Athletic	Consulting	8.5
4	Bricks and Mortar, Inc.	Consulting	8.5
5	Smith & Smith	In-house	5.5
6	Virtual Co.	In-house	2.5
7	Sweat Shops Athletic	Contract	10.5
8	IPO.com	Consulting	5.5
9	Sweat Shops Athletic	Consulting	10.0
10	Bricks and Mortar, Inc.	Consulting	6.0

Figure 18.2

Using Corporate CSS file

Obs	CustomerName	JobType	Hours
1	Smith & Smith	Consulting	8.0
2	Sweat Shops Athletic	Consulting	4.0
3	Sweat Shops Athletic	Consulting	8.5
4	Bricks and Mortar, Inc	Consulting	8.5
5	Smith & Smith	In-house	5.5
6	Virtual Co.	In-house	2.5
7	Sweat Shops Athletic	Contract	10.5
8	IPO.com	Consulting	5.5
9	Sweat Shops Athletic	Consulting	10.0
10	Bricks and Mortar, Inc	Consulting	6.0

Figure 18.3

Using Corporate CSS file

Obs	CustomerName	JobType	Hours
1	Smith & Smith	Consulting	8.0
2	Sweat Shops Athletic	Consulting	4.0
3	Sweat Shops Athletic	Consulting	8.5
4	Bricks and Mortar, Inc.	Consulting	8.5
5	Smith & Smith	In-house	5.5
6	Virtual Co.	In-house	2.5
7	Sweat Shops Athletic	Contract	10.5
8	IPO.com	Consulting	5.5
9	Sweat Shops Athletic	Consulting	10.0
10	Bricks and Mortar, Inc.	Consulting	6.0

Creating a Style Template to Match Corporate Style Guidelines

You can see from the different outputs in the previous example that the CSS style properties are rendered a bit differently by each destination. The one notable similarity in all three outputs is that the SAS title does display in a bigger, bold font than that of the table data, as we would like.

The reason for this is that the corporate CSS file provides table formatting style properties, but it does not have a CSS style property for formatting the SAS system title. ODS expects to see a .SystemTitle class selector in the CSS file. The CSS file in the previous example does not have a .SystemTitle class selector.

One way to change the style of the title in the previous example would be to edit the CSS file to have a .SystemTitle class selector. But the argument against this approach is that eventually you'd have to discover, through trial and error, the CSS selectors your procedures need. Eventually you'd end up adding almost all the SAS selectors into your CSS file. Why not let ODS do that work for you?

You can define a style template with a parent template, and then import the CSS file to modify the template to match the corporate style. In the following code, we define a new template, Styles.NewCorp, with a parent template of Styles.Printer. This ensures that all the style elements needed by ODS are available, and the bigger, bold title font from Syles.Printer will be used. Then the IMPORT command is used to bring in the corporate style CSS, so the corporate style properties will be used for the same named style elements in the new style template definition.

```
ODS PATH work.tmp(update) sashelp.tmplmst(read);

PROC TEMPLATE;
  DEFINE STYLE Styles.NewCorp;
    PARENT=Styles.Printer;
    import 'corp.css';
  END;
RUN;

ODS HTML FILE='usecorp1_18_4.HTML' STYLE=Styles.NewCorp;
ODS PDF  FILE='usecorp_18_5.PDF'  STYLE=Styles.NewCorp;
ODS RTF  FILE='usecorp_18_6.RTF'  STYLE=Styles.NewCorp;
title 'Using Corporate Style NEWCORP';
```

```
proc print data=billings(obs=10);
  var CustomerName JobType Hours;
run;
ODS _ALL_ CLOSE;
```

The output from each destination above is shown in Figures 18.4, 18.5 and 18.6

So there are two ways to use CSS files. You can reference them with the CSSSTYLE= option, or import them into a new style template. If your Web site administrator keeps the Web server locked down, then the import method might be preferable to having a separate CSS file stored on the server. On the other hand, if you have a server directory for your own use, you might want to keep all your style information in a CSS file. Both approaches are equally valid, and there are other ways that you could approach this issue of corporate style, as we'll see in upcoming examples.

Figure 18.4

Using Corporate Style Template NEWCORP

Obs	CustomerName	JobType	Hours
1	Sweat Shops Athletic	Consulting	4.0
2	Smith & Smith	Consulting	10.0
3	Smith & Smith	In-house	3.5
4	Smith & Smith	Consulting	8.5
5	Virtual Co.	Consulting	7.0

Figure 18.5

Using Corporate Style Template NEWCORP

Obs	CustomerName	JobType	Hours
1	Sweat Shops Athletic	Consulting	4.0
2	Smith & Smith	Consulting	10.0
3	Smith & Smith	In-house	3.5
4	Smith & Smith	Consulting	8.5
5	Virtual Co.	Consulting	7.0

Figure 18.6

Using Corporate Style Template NEWCORP

Obs	CustomerName	JobType	Hours
1	Sweat Shops Athletic	Consulting	4.0
2	Smith & Smith	Consulting	10.0
3	Smith & Smith	In-house	3.5
4	Smith & Smith	Consulting	8.5
5	Virtual Co.	Consulting	7.0

Using the CSSSTYLE= Option with @media Style Properties

The previous examples showed how to import a corporate stylesheet into a SAS style template. However, your company might have different style requirements for on-screen reports versus printed reports. One advantage of CSS files that you do *not* find in a SAS style template is the ability to define an @media style section in your CSS file.

An @media section is a CSS standard for how to apply different styles for different media, such as on the screen, on paper, on a Braille device or reader, and so on. The @media section is useful because on the screen you might want to have one set of colors or font properties, whereas on paper you want plain black print on white paper. For the screen, you might use a sans-serif font such as Arial, whereas on paper you prefer a serif font such as Times. The following CSS file, USE_MEDIA.CSS, shows the use of an @media print section:

```
.body {background-color: white;}
@media screen {
.header, .rowheader, .footer, .rowfooter,
  {color: white; background-color: green; font-weight:bold;
   font-size: large; font-family: Arial, sans-serif;}
.data {color: black; background-color: white;
   font-family: Arial, sans-serif; font-size: medium;}
.table {background-color: green;
   border-spacing: 1;  border: 1px black solid;
   font-size: medium; font-family: Arial, sans-serif; }
.systemtitle {font-size: large; font-family: Arial, sans-serif;
   font-weight: bold; color:green;}
}
@media print {
.header, .rowheader, .footer, .rowfooter
  {color: white; background-color: blue; font-weight:bold;
   font-size: small; font-family: Courier, Times, serif;}
.data {color: black; background-color: white;
   font-size: small; font-family: Courier, Times, serif;}
.table {background-color: blue;
   border-spacing: 1; border: 1px black solid;
   font-size: small; font-family: Courier, Times, serif;}
 .systemtitle {font-size: medium; font-family: Courier, Times, serif;
   font-weight: bold; color:blue;}
}
```

With this CSS file, we created six outputs, two for each destination. The ODS invocation statement is shown above each screen shot. Three invocations do not specify a media section to use. In this case, as we see in Figures 18.7, 18.9 and 18.11, ODS uses the SCREEN media section as the default when you do not specify a media type.

However, look at the other three outputs, 18.8, 18.10 and 18.12, which were created by specifying the PRINT media type for the CSSSTYLE= option. You can see that the font is Courier and is smaller, and that other output portions (headers, title) are blue, which is specified in the PRINT media section.

Figure 18.7

```
ods html file="c18_7.html"
    cssstyle='use_media.css';
```

CSSSTYLE USE_MEDIA with @MEDIA Sect

Obs	CustomerName	JobType	Hours
1	Sweat Shops Athletic	Consulting	4.0
2	Smith & Smith	Consulting	10.0
3	Smith & Smith	In-house	3.5
4	Smith & Smith	Consulting	8.5
5	Virtual Co.	Consulting	7.0

Figure 18.8

```
ods html file="c18_8.html"
    cssstyle='use_media.css'(print);
```

CSSSTYLE USE_MEDIA with @MEDIA Sectio

Obs	CustomerName	JobType	Hours
1	Sweat Shops Athletic	Consulting	4.0
2	Smith & Smith	Consulting	10.0
3	Smith & Smith	In-house	3.5
4	Smith & Smith	Consulting	8.5
5	Virtual Co.	Consulting	7.0

Figure 18.9

```
ods rtf file="c18_9.rtf"
    cssstyle='use_media.css';
```

CSSSTYLE USE_MEDIA with @MEDIA Sect

Obs	CustomerName	JobType	Hours
1	Sweat Shops Athletic	Consulting	4.0
2	Smith & Smith	Consulting	10.0
3	Smith & Smith	In-house	3.5
4	Smith & Smith	Consulting	8.5
5	Virtual Co.	Consulting	7.0

Figure 18.10

```
ods rtf file="c18_10.rtf"
    cssstyle='use_media.css'(print);
```

CSSSTYLE USE_MEDIA with @MEDIA Section

Obs	CustomerName	JobType	Hours
1	Sweat Shops Athletic	Consulting	4.0
2	Smith & Smith	Consulting	10.0
3	Smith & Smith	In-house	3.5
4	Smith & Smith	Consulting	8.5
5	Virtual Co.	Consulting	7.0

Figure 18.11

```
ods pdf file="c18_11.pdf"
    cssstyle='use_media.css';
```

CSSSTYLE USE_MEDIA with @MEDIA Sect

Obs	CustomerName	JobType	Hours
1	Sweat Shops Athletic	Consulting	4.0
2	Smith & Smith	Consulting	10.0
3	Smith & Smith	In-house	3.5
4	Smith & Smith	Consulting	8.5
5	Virtual Co.	Consulting	7.0

Figure 18.12

```
ods pdf file="c18_12.pdf"
    cssstyle='use_media.css'(print);
```

CSSSTYLE USE_MEDIA with @MEDIA Sectic

Obs	CustomerName	JobType	Hours
1	Sweat Shops Athletic	Consulting	4.0
2	Smith & Smith	Consulting	10.0
3	Smith & Smith	In-house	3.5
4	Smith & Smith	Consulting	8.5
5	Virtual Co.	Consulting	7.0

Using Multiple CSS Files with ODS HTML

One way to use CSS files to produce HTML output is to take advantage of the style sheet *cascade*. Cascade refers to the resolution behavior when a style element is specified multiple times in a stylesheet. During resolution of multiple style elements (known as the cascade), property values are resolved to internal or computed style values. Some of these values are inherited from a parent style element. The W3C consortium has defined how the cascade is supposed to work in resolving multiple references to HTML elements or class selectors, especially when styles of the same name are found multiple times. There are four priority levels, with 1 being the lowest priority and 4 the highest priority. Table 18.1 lists the four levels of priority and the method of specifying the style associated with each priority level.

Table 18.1: Cascading Stylesheet Priority Levels for Multiple References to HTML Elements

Priority Level	Method	Example
1	Browser default	`Paragraphs are left aligned by default`
2	External style sheet	`<link rel="stylesheet" type="text/css" href="file.css">`

3	Internal style sheet (in the <HEAD> section)	`<style type="text/css">`
		`<!--`
		`P{text-align: right;}`
		`-->`
		`</style>`

4	Inline style (an attribute in an HTML tag)	`<p style="text-align: center;">`

How can you use this to your advantage? What if your Web site administrator tells you that he wants to keep your SAS style CSS file separate from the corporate CSS file so that any corporate image references are kept separate from SAS style information? You could have one main SAS CSS file, and then have a second CSS file with the following reference to an image for the .Body class selector. The CSS file might or might not have other style settings.

```
.Body{
  margin-left: 8px; margin-right: 8px;
  background: url(fractals_inc.jpg) no-repeat top right;}
```

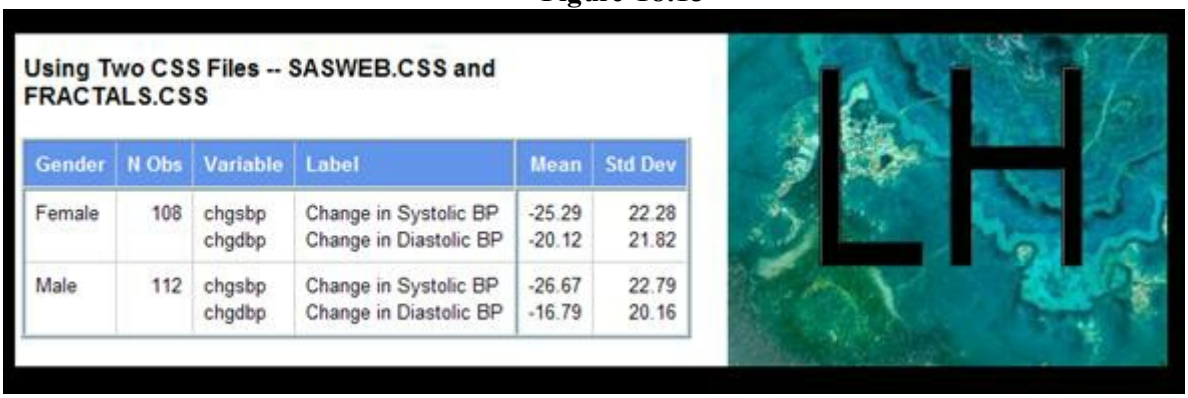
Assume that the CSS file with limited selectors is called FRACTALS.CSS and that the main SAS CSS file is called SASWEB.CSS. Both files are stored on a company Web server. You can specify both CSS files by enclosing their names in parentheses as shown in the following STYLESHEET= option.

```
ODS HTML BODY='use_two_css.html'
   STYLESHEET=(URL='sasweb.css fractals.css');
** PROC MEANS step;
ODS HTML CLOSE;
```

This would cause ODS to build two <LINK> tags, and when the browser resolved the style cascade, it would use the .Body style properties (and any others it found) in FRACTALS.CSS (combined with SASWEB.CSS) to produce the output shown in Figure 18.13. The full text of FRACTALS.CSS is shown in Figure 18.14. Note that although the browser used the image and title-related style properties, it did not use the .MyHeader class

selector that appears in the FRACTALS.CSS file.

Figure 18.13



Using Two CSS Files -- SASWEB.CSS and FRACTALS.CSS

Gender	N Obs	Variable	Label	Mean	Std Dev
Female	108	chgsbp	Change in Systolic BP	-25.29	22.28
		chgdbp	Change in Diastolic BP	-20.12	21.82
Male	112	chgsbp	Change in Systolic BP	-26.67	22.79
		chgdbp	Change in Diastolic BP	-16.79	20.16

Figure 18.14: FRACTALS.CSS

```
.Body{
  margin-left: 8px;
  margin-right: 8px;
  background: url(fractals_inc.jpg) no-repeat top right;}
.MyHeader{
  font-family: Times New Roman, Times Roman, Courier New, serif;
  font-size: medium;
  font-weight: bold;
  font-style: normal;
  color: #FFFFFF;
  background-color: #000000;}
.sysTitleAndFooterContainer{
  font-family: Arial, Helvetica, sans-serif;
  font-size: x-small;
  font-weight: normal;
  font-style: normal;
  border-width: 0;
  width: 50%;
  text-align: left;}
.SystemTitle{
    font-family: Arial, Helvetica, sans-serif;
    font-size: small;
    font-weight: bold;
    font-style: normal;
    color: #000000;}
```

Using Custom Class Selectors with ODS HTML

As we saw in the previous example, the special class selector .MyHeader in the FRACTALS.CSS file specifies a white foreground color and black background color for a header cell. Nothing in our previous example, however, caused ODS to point to the .MyHeader class selector, so we didn't see any headers with a black background in Figure 18.13.

When ODS HTML builds an HTML table definition for your output, it uses HTML class selectors to link the output to a set of CSS style properties. Here are some examples of the

type of class selectors that ODS builds:

```
<table class="Table">
<th class="l b Header" scope="col">Gender</th>
<th class="r b Header" scope="col">N Obs</th>
<th class="l t Data">Male</th>
<th class="r t Data">112</th>
```

Class selectors represent a standard way that HTML designers can link content to style properties. The ODS class selectors, Table and Header, were defined by the ODS developers to correspond to the SAS style elements, whereas other class selectors, such as L and R, were defined to ensure that proper left and right alignment happens for character and numeric values.

When the browser is rendering the following PROC PRINT report, it looks for matching CSS style properties. Assume that we want the headers for variables TX and AGE to have black backgrounds. To get the browser to apply the .MyHeader class selector to TX and AGE, we have to instruct ODS to build a class="MyHeader" attribute for these variables.

The HTMLCLASS= style attribute inserts the HTML class attribute. This attribute can be specified either in a style template (where it would affect all the headers in the report) or in a PROC PRINT, PROC REPORT, or PROC TABULATE step as shown in the following code. The VAR statements for TX and for AGE specify “MyHeader” as the value for the HTMLCLASS= style attribute. Even though the CSS file uses a “.” in front of MyHeader, you do not need to use the “.” in the HTMLCLASS= value.

```
ODS HTML BODY='use_css_selector.html'
   STYLESHEET=(URL='sasweb.css fractals.css');
proc print data=clinical(obs=5) noobs label
  style(table)={just=left};
  title 'Using Two CSS Files -- SASWEB and FRACTALS CSS';
  var tx / style(header)={HTMLCLASS="MyHeader"};
  var basedbp basesbp chgdbp chgsbp;
  var age / style(header)={HTMLCLASS="MyHeader"};
run;
ODS _ALL_ CLOSE;
```

Figure 18.15 shows the results of the PROC PRINT step. The first and last columns are highlighted differently than the other columns. Because custom class selectors do not work with RTF or PDF destinations, we would have to use the STYLE= override techniques in Chapter 19 for those destinations. The purpose of this example is to show how you would point to a class selector in a scenario where you had two CSS files.

For more information about CSS, the CSS specifications can be found at the W3C.ORG site. You will find more information about the HTMLCLASS= and HTMLSTYLE= attributes in the PROC TEMPLATE documentation on defining style elements.

Figure 18.15

Using Two CSS Files -- SASWEB.CSS and FRACTALS CSS

Treatment Group	Baseline Diastolic BP	Baseline Systolic BP	Change in Diastolic BP	Change in Systolic BP	Age in Years
1	75	113	-13	-12	25
2	87	107	2	22	28
2	90	125	-20	-18	41
1	100	171	-21	-58	60
1	108	159	-22	-40	27

Creating a Custom Style for Use with a Custom Table Template

In Chapter 10, we create some custom table templates for use with a DATA step program. One reason for creating a custom style is to link it to a column in a custom table template. Suppose you have this data, which projects the growth rate of billing information:

Division	Billable Amt_Base	Grow_ At_05	Grow_ At_10	Grow_ At_25	Grow_ At_50
Analysis	44372.55	46591.18	48809.81	55465.69	66558.83
Applications	50869.29	53412.75	55956.22	63586.61	76303.94
Documentation	46970.17	49318.68	51667.19	58712.71	70455.26
QA	50069.78	52573.27	55076.76	62587.22	75104.67
Reporting	43575.22	45753.98	47932.74	54469.03	65362.83
Systems	53446.35	56118.67	58790.99	66807.94	80169.53

Then, using techniques covered in Chapter 14, we can design custom style template Projbill_Colors and custom table template Projbill_Table as shown below.

```
/** STYLE Template **/
proc template;
  define style styles.projbill_colors;
    parent=styles.sasweb;
    class Hilite /
              background=yellow
              font_weight=bold
              foreground=black;
    class Hooray from Data/
              background=cyan
              font_weight=bold
              foreground=black;
  end;
run;
```

```
/** TABLE Template **/
proc template;
  define table projbill_table;
    column Division BillAmt;
    dynamic treatcol;
    header RepHdr;
    define RepHdr;
      text '/Projected Billing Report';
    end;
    define Division;
      style=Header;
    end;
    define BillAmt;
      define header BillHdr;
        text _LABEL_;
        style=Header;
      end;
      generic=on;
      format=dollar14.2;
      header=BillHdr;
      style=treatcol;
    end;
  end;
run;
```

The Projbill_Colors style template has Sasweb as its parent, and it includes two new style elements, Hilite and Hooray. The custom table template Projbill_Table defines a generic report column called BILLAMT. When BILLAMT is referenced, its style is assigned dynamically by the value of dynamic variable TREATCOL. Code in the DATA step program

that we use to create a report assigns values to TREATCOL.

```
ODS LISTING CLOSE;
ODS RTF FILE='Special_Style.rtf' STYLE=styles.projbill_colors;
ODS PDF FILE='Special_Style.pdf' STYLE=styles.projbill_colors;
ODS HTML FILE='Special_Style.html' STYLE=styles.projbill_colors;
title 'Using Custom Table and Style Template';

data _null_;
set projected_billing;
file print ods=(template='projbill_table'
  columns=(Division
      BillAmt=BillableAmt_Base(generic=on dynamic=(treatcol='Hilite'))
          BillAmt=Grow_At_05(generic=on dynamic=(treatcol='Data'))
          BillAmt=Grow_At_25(generic=on dynamic=(treatcol='Hooray'))));
put _ods_;
label Grow_At_05 = 'Growth at 5%'
      Grow_At_50 = 'Growth at 50%'
      BillableAmt_Base = 'Baseline (Actual)';
run;
ODS _ALL_ CLOSE;
```

The three ODS invocation statements reference the style template Styles.Projbill_Colors. The FILE PRINT ODS statement references table template Projbill_Table. The Hilite style element is used for the BILLABLEAMT_BASE variable column in the report. The regular Data style element, which is found in Styles.Sasweb, is used for the GROW_AT_05 variable column. The Hooray style element is specified as the value of the TREATCOL dynamic item for the GROW_AT_25 report column.

If you examine Figure 18.16 (the HTML output), you will see that the new style elements are used for the columns as specified. The RTF and PDF outputs look essentially the same as the HTML output, so they are not shown. Remember that you could just as easily use a custom style template and custom style elements with PROC REPORT, PROC PRINT or PROC TABULATE instead of a DATA step and custom table template. Or you could design a custom style template completely from scratch. However, the style templates defined by SAS account for all the possible style elements that any SAS procedure can request, so designing a style

template completely from scratch is generally more effort than it is worth.

Figure 18.16

Using Custom Table and Style Template

Projected Billing Report			
Division	Baseline (Actual)	Growth at 5%	Growth at 25%
Analysis	$44,372.55	$46,591.18	$55,465.69
Applications	$50,869.29	$53,412.75	$63,586.61
Documentation	$46,970.17	$49,318.68	$58,712.71
QA	$50,069.78	$52,573.27	$62,587.22
Reporting	$43,575.22	$45,753.98	$54,469.03
Systems	$53,446.35	$56,118.67	$66,807.94

Using Style Templates to Make Smaller Output Files

We've talked in previous chapters about using COMPRESS= to reduce the size of PDF files and of using PHTML or CHTML to reduce the size of HTML files. You might also want to have your output take up less "real estate" on the page— that is, reduce the width and/or height of the report output. ODS uses two main style settings to control the overall size of ODS output: the font sizes and the cell padding around values in cells. The cell padding is the space between cell text and the rules on all four sides of the text. For example, the following PROC TEMPLATE step defines style Styles.Smaller, which shrinks the output to about 80% of the usual size by reducing font sizes and cell padding. The parent template of Styles.Smaller is Styles.Printer.

```
proc template;
  define style styles.smaller;
    parent = styles.printer;
    class fonts /
      'titlefont2' = ("Times",10pt,Bold Italic)
      'titlefont' = ("Times",11pt,Bold Italic)
      'strongfont' = ("Times",8pt,Bold)
      'emphasisfont' = ("Times",8pt,Italic)
      'fixedEmphasisfont' = ("Courier New, Courier",7pt,Italic)
      'fixedStrongfont' = ("Courier New, Courier",7pt,Bold)
      'fixedHeadingfont' = ("Courier New, Courier",7pt,Bold)
      'batchfixedfont' = ("SAS Monospace, Courier New, Courier",5pt)
      'fixedfont' = ("Courier New, Courier",7pt)
      'headingemphasisfont' = ("Times",9pt,Bold Italic)
      'headingfont' = ("Times",9pt,Bold)
      'docFont' = ("Times",8pt);
    class table /
      rules = all
      cellpadding = 2pt
      cellspacing = 0;
```

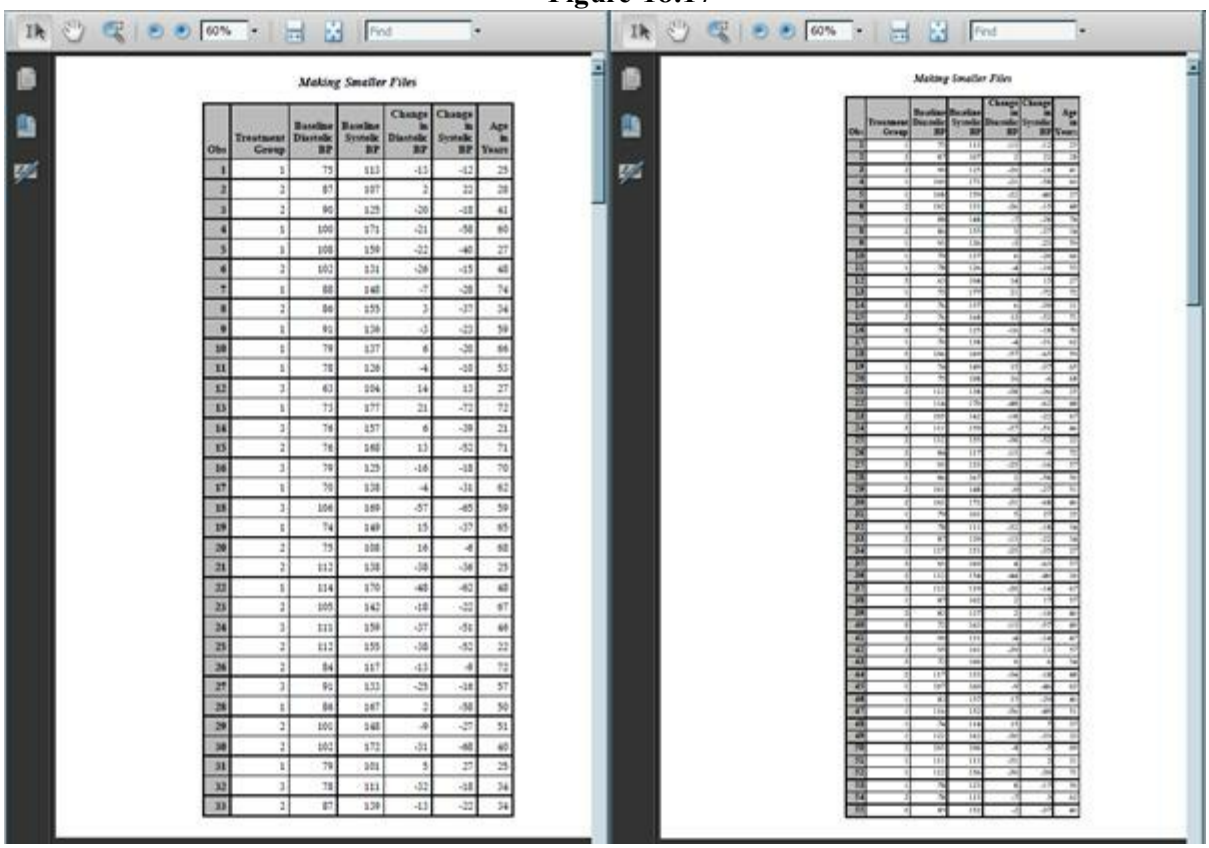
```
  end;
run;
```

The result of changing just the font sizes and cell padding is dramatic and can be seen in Figure 18.17. Only 33 observations fit on a page in the first PDF file on the left, whereas 55 observations fit on a page in the second PDF file on the right. The total page count in the PDF files went from seven to four. Here is the code that produced Figure 18.17:

```
ODS PDF(1) FILE='standard.pdf';
ODS PDF(2) FILE='smaller.pdf' STYLE=Styles.Smaller;
  proc print data=clinical label;

    var tx basedbp basesbp chgdbp chgsbp age;

  run;

ODS _ALL_ CLOSE;
```

For RTF, you should see an equivalent reduction in number of pages, depending on the word processor margin settings. Of course, if you use the style template with HTML, the output itself will be smaller, but the page breaks will be controlled by the browser's printing.

Figure 18.17



Inserting an Image with a Style Template

In Chapter 4 and Chapter 5, we showed how to insert an image into either an RTF or PDF file using ODS ESCAPECHAR syntax. Another way to insert an image into your output is with a style template. The PREIMAGE= (or POSTIMAGE=) style attribute would be used for the style element of interest. You could associate the image with a title, footnote, header,

or even a data cell. In our template, designed for ODS PDF, the image is associated with the "body" style element, to ensure that it appears only once in the document.

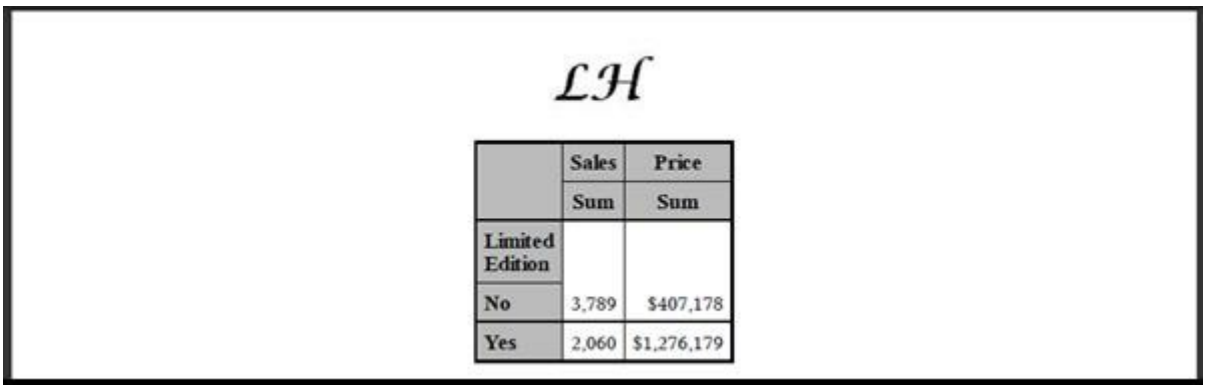
```
proc template;
   define style printerlogo;
   parent=styles.printer;
   style body from body /
      preimage='c:\books\ods\logo.jpg';
   end;
run;

ods pdf file="c:\temp\EditionSales_18_18.pdf"
      STYLE=Printerlogo startpage=no;
proc tabulate data=gallery;
  class limited;
  var sales price;
  table limited, sales*sum*f=comma6. price*sum*f=dollar12.;
run;
ods _all_ close;
```

The output from the preceding program is shown in Figure 18.18. Remember that images are converted to internal PDF (or RTF) format at the time the output file is created. So the location of the image, for the PREIMAGE= attribute, needs to be specified in a physical location that is available on the system or server when the output file is created. If you wanted to see the logo at the top of every page, you would only need to have a style element definition for the SystemTitle.

```
proc template;
   define style printerlogo2;
   parent=styles.printer;
   class systemTitle /
      just=r
      preimage='c:\books\ods\logo.jpg';
   end;
run;

ODS PDF FILE="c:\temp\EditionSales.pdf"
      STYLE=PrinterLogo2 startpage=no;
proc tabulate data=gallery;
  title ' ';
  . . . same TABLE statement . . .
run;
ODS _ALL_ CLOSE;
```

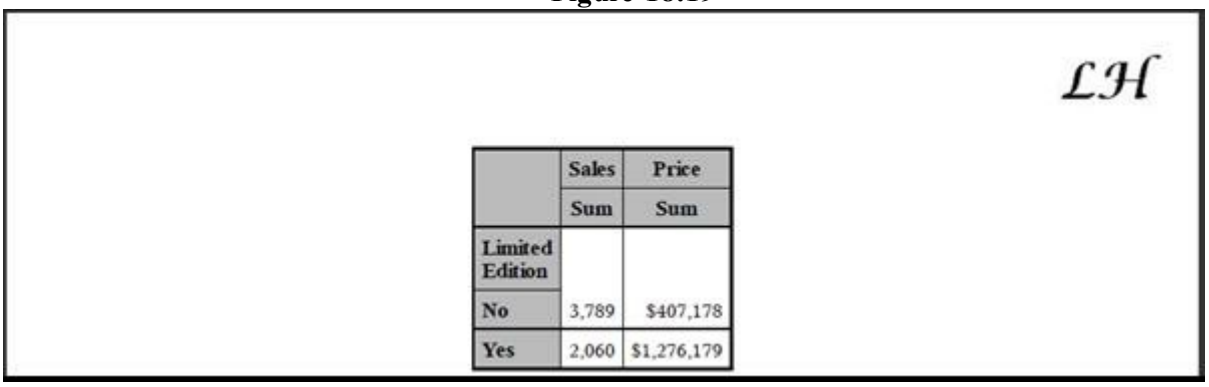
The first page of the output rendered by the revised template is shown in Figure 18.19. Note that this second template actually requires a TITLE statement in order for the logo to be placed in the output. Without a TITLE statement, no logo would appear. This same technique would work for RTF and HTML destinations as well. However, for the HTML destination, the location of the image file would need to be changed to the location of the image when the HTML file is opened, as opposed to the location when the PDF (or RTF) file is created.

Figure 18.18



	Sales	Price
	Sum	Sum
Limited Edition		
No	3,789	$407,178
Yes	2,060	$1,276,179

Figure 18.19



	Sales	Price
	Sum	Sum
Limited Edition		
No	3,789	$407,178
Yes	2,060	$1,276,179

Using RTF Control Strings with a Style Template

In Chapter 4, we showed how to use RTF control strings or field codes in the creation of RTF output. One frequent requirement when producing academic reports is to create the type of report where a line appears above the first footnote on the page.

In SAS 9.1.3 and SAS 9.2, this can be accomplished using ODS ESCAPECHAR syntax with the PRETEXT= attribute. In SAS 9.2, the {STYLE} form of the ODS ESCAPECHAR syntax does not work. The relevant RTF control string for altering a cell border is as follows:

```
\brdrt\brdrs\brdrw1
```

If you were developing reports for production use, you would not want to always put the RTF control string into your FOOTNOTE statement. It would make more sense to put the RTF control string, as the value of the PRETEXT= attribute, into a style template that you would always use for this type of report.

Another way to insert an image into your output is with a style template. The PREIMAGE= (or POSTIMAGE=) style attribute would be used for the style element of interest. You could associate the image with a title, footnote, header, or even a data cell. In our template, designed for ODS RTF, the image is associated with the “body” style element, to ensure that it only appears once in the document.

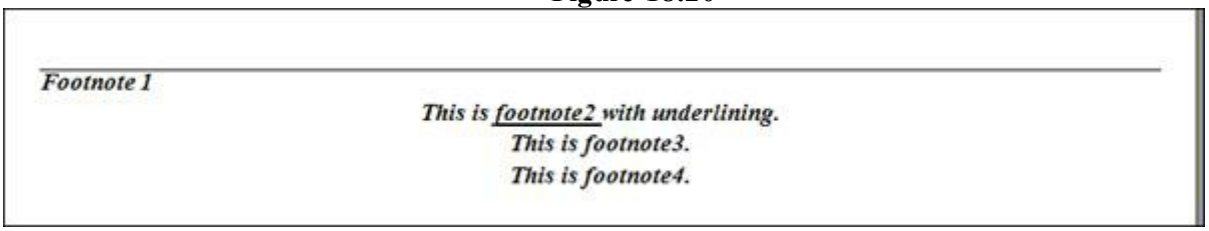
```
proc template;
  define style styles.usefc;
  parent=styles.rtf;
    class systemfooter  /
      protectspecialchars=off
```

```
      pretext='\brdrt\brdrs\brdrw1 ';
    class systemfooter2, systemfooter3, systemfooter4/
      pretext=_undef_;
  end;
run;
ODS RTF FILE='Payroll_Title.rtf' style=styles.usefc;
title "Payroll Report";
footnote j=l "Footnote 1";
footnote2 "This is \ul footnote2 \ul0 with underlining.";
footnote3 "This is footnote3.";
footnote4 "This is footnote4.";
proc means data=hr nonobs maxdec=0 mean sum;
   class Department;
   var AnnualSalary;
run;
ODS RTF CLOSE;
```

The output from the preceding program is shown in Figure 18.20. The PRETEXT= attribute uses the RTF control string in quotes (with a space following the \brdrw1 string). Because the backslash needs to have protection turned off, the PROTECTSPECIALCHARS attribute is set for the SystemFooter element too. Because we want to have other FOOTNOTE statements in our program, we use the SAS 9.2 form of the CLASS statement to turn off the PRETEXT= attribute (using _UNDEF_ as the attribute value), so that the other footnotes (FOOTNOTE2 through FOOTNOTE4) do not inherit the PRETEXT= value.

Figure 18.20 shows in Print Preview mode only the bottom of the page of the results. Note how the \ul control string was able to be used in FOOTNOTE2. Because the other footnotes inherit their attributes from the SystemFooter element, the protection of special characters was still turned off, which allowed the \ul control string to be passed to the RTF file as a control string.

Figure 18.20



Storing and Retrieving a Personal or Group Template

The ODS PATH statement enables you to control the storage and retrieval path for updated templates. For example, when you test your PROC TEMPLATE code, you might want to write the templates to the WORK library. If you want to create a template for your personal use, you can use the ODS PATH statement to control where the template is stored, such as in a personal item store for templates.

When your Base SAS session starts, SAS by default automatically submits the following ODS PATH statement. This statement specifies that SAS access the two item stores in the

order in which they are listed when storing and retrieving templates.

```
ODS PATH sasuser.templat(update) sashelp.tmplmst(read);
```

You can also add item stores to the list on the ODS PATH statement. You add them in the order you want ODS to access them when it stores and retrieves your templates. For example, the following ODS statement starts its search in personal item store MYLIB.MYTEMPLATES, where MYLIB points to a location defined by a LIBNAME statement you specify prior to ODS PATH statement.

```
ODS PATH mylib.mytemplates(update)
    sasuser.templat(update)
    sashelp.tmplmst(read);
```

The UPDATE option on the first two item stores allows Read and Write access to those item stores. The READ option on the third item store permits read-only access to that item store. SASHELP.TMPLMST contains SAS built-in style templates and is not an item store that you should modify.

Similarly, if you wanted to create a group-level item store (for example, for members of a programming group to share), you could list the group-level item store in your ODS PATH statement. Assume libref MYGRP points to a shared location, and the name of the item store that the group shares is GRPTEMPLATES. In this scenario, your ODS PATH statement would be as follows:

```
ODS PATH mygrp.grptemplates(update)
    sasuser.templat(update)
    sashelp.tmplmst(read);
```

If you include the ODS PATH statement in your AUTOEXEC.SAS file or startup script, its settings remain in effect throughout your SAS session unless you submit another ODS PATH statement during the session. When you define a group-level item store, all members of the group would have to include the ODS PATH statement in their AUTOEXEC.SAS or other startup script.

If you are working with SAS Enterprise Guide, or any other application or system that has its own template libraries, you may want to use slightly different ODS PATH syntax to put your template at the front of the search list without disturbing the ODS PATH order needed by SAS Enterprise Guide or the application. It is unlikely that you would pick a template name used by SAS Enterprise Guide, but just in case, you might want to pick template names that are unique to your group. To insert your template as a read-only item store into the search path without redefining the entire ODS PATH list, insert the PREPEND option before naming your item store as follows:

```
ODS PATH (PREPEND) mygrp.grptemplates(read);
```

There is also a corresponding APPEND option for the ODS PATH statement that works exactly as you would expect. This option appends your item store to the current ODS PATH search list. Of course, the PREPEND and APPEND options are available for any usage situation in which you want to alter the template search list, not just for use with SAS Enterprise Guide.

If you want to programmatically document the current ODS PATH list or document a change to the list, use the ODS PATH SHOW statement. Consider the following sequence of ODS PATH statements:

```
**a) Reestablish the entire path;
ODS PATH work.mytemp(update)
      sashelp.tmplmst(read);
ODS PATH SHOW;
** b) Reset the path;
ODS PATH RESET;
ODS PATH SHOW;
** c) Prepending an item store;
ODS PATH(PREPEND) work.pretemp(update);
ODS PATH SHOW;
** d) Appending an item store;
ODS PATH(APPEND) work.apptemp(update);
ODS PATH SHOW;
```

If you examine the SAS log after these statements execute, you will see that the statements work exactly as described in the documentation. Figure 18.21 shows the results of steps A through D, which illustrate adding item stores and usage of the PREPEND, APPEND, and SHOW options.

Figure 18.21

```
14   **a) Restablish the entire path;
15   ODS PATH work.mytemp(update)
16            sashelp.tmplmst(read);
17   ODS PATH SHOW;
Current ODS PATH list is:

1. WORK.MYTEMP(UPDATE)
2. SASHELP.TMPLMST(READ)
18
19   ** b) Reset the path;
20   ODS PATH RESET;
21   ODS PATH SHOW;
Current ODS PATH list is:

1. SASUSER.TEMPLAT(UPDATE)
2. SASHELP.TMPLMST(READ)
22
23   ** c) Prepending an item store;
24   ODS PATH(PREPEND) work.pretemp(update);
25   ODS PATH SHOW;
Current ODS PATH list is:

1. WORK.PRETEMP(UPDATE)
2. SASUSER.TEMPLAT(UPDATE)
3. SASHELP.TMPLMST(READ)
26
27   ** d) Appending an item store;
28   ODS PATH(APPEND) work.apptemp(update);
29   ODS PATH SHOW;
Current ODS PATH list is:

1. WORK.PRETEMP(UPDATE)
2. SASUSER.TEMPLAT(UPDATE)
3. SASHELP.TMPLMST(READ)
4. WORK.APPTEMP(UPDATE)
```

Removing Access to a Personal or Group Template

After your code runs, you have several choices for how to gracefully remove your template from the ODS PATH. You can restore the ODS PATH to the defaults, which are SASUSER.TEMPLAT and SASHELP.TMPLMST. This can be done using the following statement after your code runs:

```
ODS PATH RESET;
```

However, if you are using an application that depends on its own item store, you might interfere with the templates that the application expected to find.

To simply remove your special item store or template store from the ODS PATH search list, you can reverse the PREPEND action of the ODS PATH statement by submitting the following statement:

```
ODS PATH (REMOVE) mygrp.grptemplates(read);
```

This syntax removes your item store from the ODS PATH list and leaves the other items stores in whatever order they were originally set.

Again, you can use the ODS PATH SHOW statement to programmatically document your changes:

```
** e) Removing an item store;

ODS PATH(REMOVE) work.apptemp(update);

ODS PATH SHOW;


** f) Reset the path;

ODS PATH RESET;

ODS PATH SHOW;
```

Figure 18.22 shows the SAS log from steps E and F, which illustrate removing item stores and using the REMOVE and RESET options.

When you start working with ODS template definitions, you might not initially bother with issuing an ODS PATH statement. That is an acceptable approach because SAS expects that SASHELP.TMPLMST will be defined as a read-only item store on your system. If you attempt to modify or write a custom template definition (of any kind), by default ODS stores the template definition in the SASUSER.TEMPLAT item store.

However, in a shared network installation of SAS, it is likely that every user will have his or her own SASUSER library, so one person's SASUSER item store would not be available to another person in the same group. This is the kind of scenario where you would use the ODS PATH statement to define a search and retrieval path for your template stores instead of taking the ODS default.

It might make sense for you to have one template store for all your style templates and another for tagset templates or graph templates. It might be more logical for you to organize all your changed templates into one item store. When you do modify or update a template, however, be sure to save your PROC TEMPLATE code. Template stores or item stores are not portable across operating systems, so you will need to re-create the templates by rerunning the code on different operating systems.

Figure 18.22

```
30
31    ** e) Removing an item store;
32    ODS PATH(REMOVE) work.apptemp(update);
33    ODS PATH SHOW;
Current ODS PATH list is:

1. WORK.PRETEMP(UPDATE)
2. SASUSER.TEMPLAT(UPDATE)
3. SASHELP.TMPLMST(READ)
34
35    ** f) Reset the path;
36    ODS PATH RESET;
37    ODS PATH SHOW;
Current ODS PATH list is:

1. SASUSER.TEMPLAT(UPDATE)
2. SASHELP.TMPLMST(READ)
```

Specifying an Alternate Template as a Backup

One thing you might want to do is check for the existence of a template, such as a style template. Then, if the style template that you want is not available, you can use a backup style for your ODS job. To simply find out whether your choice of style template exists, you can use a dictionary table or a SASHELP view. A more complicated approach might involve using the EXIST and OPEN functions in a macro program to determine whether your item store exists and/or issuing an ODS PATH statement. However, we want to focus on just testing the existence of a style template and issuing a backup template name.

New with SAS 9.2 is a DICTIONARY file called DICTIONARY.STYLES, which has a parallel view called SASHELP.VSTYLE. Let's say that you choose a style called MYSTYLE for your ODS output. If ODS cannot find that style in the available template stores, it uses the default style template for your destination. However, you want to be able to specify something other than the default template if your style template is not found.

Consider the following program, which checks if a style exists, uses it if it does, and uses a different style if it does not. The program uses macro variables defined at the beginning of the program with %LET statements and macro variables assigned values by a PROC SQL step that examines DICTIONARY.STYLES for the existence of the style.

The program initially creates five macro variables.

1. The value assigned to the macro variable CHOICE specifies the style template we want to use as our first choice. This example specifies the style template Analysis as our first choice.
2. The value assigned to macro variable USESTYLE is the style template we want to use if our first choice style template, Analysis, is not available. This example specifies SAS built-in style template Sasweb as our second choice if Analysis is not available.
3. The value assigned to macro variable ITM tells us whether the template is available. It is initialized to the text string "notfound".
4. Macro variable ILIB is defined and initialized to null before it is used in the PROC SQL step. It will hold the libref of the style that PROC SQL selects in its query.
5. Macro variable IMEM is defined and initialized to null before it is used in the PROC SQL step. It will hold the member name of the style that PROC SQL selects in its query.

The PROC SQL step queries three columns in DICTIONARY.STYLES and assigns the values it extracts to three macro variables: ILIB, IMEM, and USESTYLE. The WHERE clause restricts the query to rows in DICTIONARY.STYLES where the value for column STYLE is equal to the value assigned to macro variable CHOICE.

When the query selects a row from DICTIONARY.STYLES that satisfies the WHERE clause, PROC SQL assigns values to the three macro variables on the SELECT statement. In that situation, PROC SQL replaces the value in the macro variable USESTYLE with the name of this first-choice style.

When the query does not select a row from DICTIONARY.STYLES, PROC SQL does not assign values to the three macro variables, and the value of the macro variable USESTYLE remains unchanged. If PROC SQL finds the Analysis style in DICTIONARY.STYLES, it assigns ANALYSIS to macro variable USESTYLE. If PROC SQL does not find the Analysis style in DICTIONARY.STYLES, the value of USESTYLE remains Sasweb.

The STYLE= option of the ODS HTML statement specifies the macro variable USESTYLE.

The purpose of the macro variable ITM is to store the complete item store name so that it can be displayed in the SAS log with the %PUT statement. The %LET statement for ITM concatenates the values of macro variables ILIB and IMEM.

```
%let choice=analysis;

%let itm = notfound;

%let usestyle=sasweb;

%let ilib=;

%let imem=;
```

```
proc sql;
  select libname, memname, style into :ilib, :imem, :usestyle
  from dictionary.styles
  where upcase(style) contains "%upcase(&choice)";
quit;

%let itm = %sysfunc(compress(&ilib..&imem));
%let usestyle=&usestyle;
%put itm is &itm and usestyle is &usestyle;
ODS HTML FILE='teststyle.html' STYLE=&usestyle;
proc print data=billings;
run;
ODS HTML CLOSE;
```

The results of the %PUT statements and a partial SAS log are shown in Figure 18.23 for a valid style name, Analysis, and in Figure 18.24 for an invalid style name, NOTASTYLE. With option SYMBOLGEN in effect, the output underneath the ODS HTML invocation statement shows the final value of the USESTYLE macro variable. Of course, you could incorporate this type of code in a more robust macro program for production purposes.

Figure 18.23

```
130  ** Chapter 18 Testing for a style that exists;
131  %let choice=analysis;
132  %let itm = notfound;
133  %let usestyle=sasweb;
134  %let ilib=;
135  %let imem=;
136
137  proc sql;
138    select libname, memname, style into :ilib, :imem, :usestyle
139    from dictionary.styles
140    where upcase(style) contains "%upcase(&choice)";
141  quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.35 seconds
      cpu time            0.35 seconds


142
143  %let itm = %sysfunc(compress(&ilib..&imem));
144  %let usestyle=&usestyle;
145  %put itm is &itm and usestyle is &usestyle;
itm is SASHELP.TMPLMST and usestyle is Styles.Analysis
146
147  options symbolgen;
148  ODS HTML FILE='teststyle.html' STYLE=&usestyle;
SYMBOLGEN:  Macro variable USESTYLE resolves to Styles.Analysis
NOTE: Writing HTML Body file: teststyle.html
149  proc print data=billings;
150  run;
```

Figure 18.24

```
154  ** Chapter 18 Testing for a style that does not exist;
155  %let choice=notastyle;
156  %let itm = notfound;
157  %let usestyle=sasweb;
158  %let ilib=;
159  %let imem=;
160
161  proc sql;
162    select libname, memname, style into :ilib, :imem, :usestyle
163    from dictionary.styles
164    where upcase(style) contains "%upcase(&choice)";
NOTE: No rows were selected.
165  quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.35 seconds
      cpu time            0.34 seconds


166
167  %let itm = %sysfunc(compress(&ilib..&imem));
168  %let usestyle=&usestyle;
169  %put itm is &itm and usestyle is &usestyle;
itm is . and usestyle is sasweb
170
171  options symbolgen;
172  ODS HTML FILE='teststyle.html' STYLE=&usestyle;
SYMBOLGEN:  Macro variable USESTYLE resolves to sasweb
NOTE: Writing HTML Body file: teststyle.html
173  proc print data=billings;
174  run;
```

Chapter 19
Special Cases: The REPORT, TABULATE, and PRINT Procedures

For most procedures in SAS, the only way to change the appearance of the output is to change or modify the style template definition. Three procedures are the exception: REPORT, TABULATE, and PRINT. These procedures allow you to change the output style attributes on the fly when the output is generated.

With these three procedures, you can create almost any type of tabular report. Add in the extra control over style attributes and you have a reporting powerhouse.

This chapter shows how to change fonts, colors, and alignment of your output. You will also learn how to use formats to highlight key results in special colors and use images in table headings.

For convenience, all of the examples are shown as HTML output. Except where noted, the examples all work for RTF or PDF output as well.

This chapter uses the data set FURNITURE to illustrate the concepts. This data set is available on the author pages site for this book at http://support.sas.com/authors.

For a color version of the output from this chapter, see the author pages for this book at http://support.sas.com/authors.

Changing PROC REPORT Column Heading Style Attributes

Applying style attributes to PROC REPORT output is very simple. All you have to do is add STYLE= statement-level options to whichever part of the output you want to change.

There are a number of ways you can use the STYLE= option in PROC REPORT. We start out with one simple technique: using the STYLE= option to change the column headings. We start with a table similar to the previous examples.

```
      ODS HTML BODY='furniture.html' STYLE=sasweb;
proc report data=furniture nowd;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
run;
ODS HTML CLOSE;
```

This code produces the output shown in Figure 19.1. It's a basic table using the Sasweb built-in template definition. The typeface used in the headings and table body is Arial. As an example, we will change this typeface to Times New Roman.

This is done by adding a STYLE= option to the PROC REPORT statement. The STYLE keyword is followed by the name of the location we want to modify. This location name is enclosed in parentheses. In this case, the location is called Header. A list of the locations that you can modify is in the SAS OnlineDoc for the REPORT procedure.

Follow the name of the location with an equal sign and the list of the style attributes you want to change. You can list one or more attributes. The attributes must be enclosed in square brackets [ ] or curly brackets { }. Details about these attributes and their settings can be found in the SAS OnlineDoc.

The following code makes two changes: the font typeface is changed to Times New Roman and the background of the headers is changed to gray. Notice the use of quotation marks around the typeface name. Because "Times New Roman" contains spaces, these quotation marks are necessary.

```
      ODS HTML BODY='furntiure2.html' STYLE=sasweb;
proc report data=furniture nowd
      STYLE(Header)=[FONT_FACE='Times New Roman' background=gray];
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
run;
ODS HTML CLOSE;
```

The results are shown in Figure 19.2. This technique can be used to change any of the column header attributes.

Figure 19.1

Retail Stock Report

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.2

Retail Stock Report

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Changing PROC REPORT Table Cell Style Attributes

In addition to being able to control style attributes for PROC REPORT column headings, you can also control style attributes for the cells under a particular heading. This example shows how to change the background color for the table cells with the dollar amounts, which are for the column PRICE.

You do this by adding a STYLE= option to the DEFINE PRICE statement. The STYLE= option can be added anywhere after the slash (/). The following code changes the white background color of the Sasweb default to a shade of gray (cxDDDDDD).

The STYLE(Header)= option in the last example applied the new style attribute to the table headers. In this example, you use the STYLE(Column)= option to apply the new style attribute to the cells of a specific column.

```
      ODS HTML BODY='stock3.html' STYLE=sasweb;
proc report data=furniture nowd;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.
    STYLE(Column)=[BACKGROUND=cxDDDDDD];
run;
ODS HTML CLOSE;
```

This code produces the output shown in Figure 19.3. You can see that the table cells with the dollar amounts now have a gray background. The column header retains its original color scheme.

If you want to change the style attribute of the column header for the PRICE column, add a STYLE(Header)= option to the DEFINE PRICE statement.

```
      ODS HTML BODY='stock4.html' STYLE=sasweb;
proc report data=furniture nowd;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.
    STYLE(Column)=[BACKGROUND=cxDDDDDD]
    STYLE(Header)=[BACKGROUND=cx336699];
run;
ODS HTML CLOSE;
```

This code produces Figure 19.4, with a darker color used for the column heading.

Retail Stock Report

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.3

Figure 19.4

Retail Stock Report

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Changing PROC REPORT Row Style Attributes

To change the style attributes of a column of a PROC REPORT table, you can use a STYLE= option in the DEFINE statement that builds the column. To change the style attributes of a row, you use a different technique. This example shows how to modify the row style attributes to create rows of alternating colors. This can make your table much easier to read.

In addition to using the STYLE= option in the DEFINE statement, you can use a CALL DEFINE statement inside a COMPUTE statement. This enables you to call up the row style element and modify its attributes. The CALL DEFINE statement has three parameters: the name of the element being modified (_ROW_), the description of what is being modified (STYLE), and the STYLE= option code to make the change.

The following code shows how to color-code the alternating rows. The COMPUTE statement contains an IF statement that selects odd numbered rows. Those rows are the ones whose background color is changed to a shade of gray.

```
      ODS HTML BODY='stock4.htm' STYLE=sasweb;
proc report data=furniture nowd;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
  compute Material;
    count+1;
    if (mod(count,2)) then do;
      CALL DEFINE(_ROW_, "STYLE", "STYLE=[BACKGROUND=cxDDDDDD]");
    end;
  endcomp;
run;
ODS HTML CLOSE;
```

The resulting table is shown in Figure 19.5. Notice how rows 1, 3, and 5 are now gray. If you want to switch this around so that the even-numbered rows are gray, modify the IF statement inside the COMPUTE block to select the even rows.

```
      if (mod(count,2)=0) then do;
   CALL DEFINE(_ROW_, "STYLE", "STYLE=[BACKGROUND=cxDDDDDD]");
end;
```

The revised code creates the table shown in Figure 19.6. Now rows 2, 4, and 6 are gray. This is a fairly complex example for this introductory text, but the technique is very useful. Even if you don’t understand all of the details of this example, you can copy the code and put it to use.

Figure 19.5

Retail Stock Report

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.6

Retail Stock Report

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Changing PROC REPORT Subtotals and Totals Style Attributes

A common technique in PROC REPORT is to create totals at the end of each group of results. PROC REPORT gives you some ability to control the format of these breaks in your listing output. You can add single or double overlines or underlines, and you can skip a row. If you want to do fancier formatting of your subtotals and totals, turn to ODS.

The following PROC REPORT step computes totals using the BREAK and RBREAK statements. It applies the Sasweb style to the output, which is shown in Figure 19.7.

```
      ODS HTML BODY='detailstock.html' STYLE=sasweb;
proc report data=furniture nowd;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
  break after Type / summarize;
  rbreak after / summarize;
run;
ODS HTML CLOSE;
```

This table has all of the results we need, but it could be made easier to read. By adding some formatting to the subtotals and totals, you can make them stand out. The STYLE = option added to the BREAK AFTER statement adds some space below each subtotal by increasing the cell height and vertically justifying the text. It also sets the font weight to bold for the subtotals. The STYLE= option in the RBREAK AFTER statement sets the font weight of the report totals to bold.

```
      ODS HTML BODY='detailstock2.html' STYLE=sasweb;
proc report data=furniture nowd;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
  break after Type / summarize STYLE=[VJUST=T CELLHEIGHT=50
                                 FONT_WEIGHT=Bold];
  rbreak after / summarize STYLE=[FONT_WEIGHT=Bold];
run;
ODS HTML CLOSE;
```

The revised table is shown in Figure 19.8.

Figure 19.7

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Coffee		*$79*
Dining	Glass/Metal	$400
	Wood	$196
Dining		*$275*
End	Glass/Metal	$104
	Wood	$59
End		*$77*
		$175

Figure 19.8

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Coffee		***$79***
Dining	Glass/Metal	$400
	Wood	$196
Dining		***$275***
End	Glass/Metal	$104
	Wood	$59
End		***$77***
		$175

Adding Traffic Lighting to PROC REPORT Table Values

What do traffic lights have to do with SAS output? The answer is that you can highlight results in your output using the familiar red, yellow, and green lights. You can use red for bad results, yellow for neutral results, and green for good results. If you're creating a large table, this technique is great for focusing the reader's attention on the key results.

In this example, we take our familiar table from the first few examples in this chapter and apply traffic lighting to mark low prices in green, high prices in red, and prices in between in yellow. This is done with a COMPUTE block on the variable PRICE. Within the COMPUTE block, a series of IF, THEN, ELSE statements assign the foreground colors based on the value of PRICE.

The three color settings are RGB values for green, red and yellow. The fonts are all changed to bold to stand out against the white background.

```
      ODS HTML BODY='trafficstock.html' STYLE=sasweb;
proc report data=furniture nowd;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
  compute Price;
    if Price.mean <= 100 then
      CALL DEFINE(_COL_,"STYLE",
          "STYLE=[FOREGROUND=cx006633 FONT_WEIGHT=Bold]");
    else if Price.mean >= 300 then
      CALL DEFINE(_COL_,"STYLE",
          "STYLE=[FOREGROUND=cxCC0000 FONT_WEIGHT=Bold]");
    else
      CALL DEFINE(_COL_,"STYLE",
          "STYLE=[FOREGROUND=cxFF9900 FONT_WEIGHT=Bold]");
  endcomp;
run;
ODS HTML CLOSE;
```

The results are shown in Figure 19.9. The prices are colored to draw attention to the high and low values. You can also use this technique in reverse by coloring the backgrounds of the cells in green, red, and yellow. To do this, remove the FOREGROUND= attribute and replace it with the BACKGROUND= attribute. Pick slightly lighter shades of the three colors so they don't overwhelm the foreground text. The colors cx00CC33 for green, cxFF3300 for red, and cxFFFF66 for yellow, are good choices. Keep the fonts in bold so that the text will stand out. This version of the table is illustrated in Figure 19.10.

Figure 19.9

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.10

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

If you are going to use this technique, you should probably also take the time to design a style definition with a color scheme that is a better match for the red, yellow, and green traffic lighting. A simpler black and white color scheme might be the best approach.

Another way to create traffic lighting in PROC REPORT is to use a user-defined format to assign the colors. The example "Adding Traffic Lighting to PROC TABULATE Table Values" later in this chapter illustrates this technique.

Adding Graphics to PROC REPORT Tables

Changing fonts and colors goes a long way toward livening up your output, and adding graphics takes your results to a new level. This example shows how to add a heading with a graphic to your PROC REPORT results.

The first thing you need to do is set up the heading. This is done with a COMPUTE BEFORE _PAGE_ statement and a LINE statement. This action adds a new row to the top of the table. The new row contains the text given in the LINE statement, as shown in Figure 19.11. Unfortunately, the row is created using the Sasweb table cell style attributes, which makes the title rather faint in appearance.

```
      ODS HTML BODY='graphic.html' STYLE=sasweb;
proc report data=furniture nowd;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
  compute before _page_ / LEFT;
    LINE "Totally Tables, Inc.";
  endcomp;
run;
ODS HTML CLOSE;
```

To fix the appearance of the title and add a graphic, you can use the STYLE= option in the COMPUTE statement. First, add the graphic using the PREIMAGE= attribute. This example assumes that the graphic is stored in the same file location as the HTML page you are creating. The second fix is to make the font bolder and bigger, and to change it to a color that coordinates with the graphic.

```
      ODS HTML BODY='graphic2.html' STYLE=sasweb;
proc report data=furniture nowd;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
  compute before _page_ / LEFT
     STYLE=[PREIMAGE='dining.gif'
          FONT_WEIGHT=Bold FONT_SIZE=5 FOREGROUND=cx993300];
    LINE "Totally Tables, Inc.";
  endcomp;
run;
ODS HTML CLOSE;
```

The new results are shown in Figure 19.12.

Figure 19.11

Totally Tables, Inc.

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.12

Totally Tables, Inc.

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Creating an Output Data Set from PROC REPORT

With other procedures, you use ODS OUTPUT to create an output data set. With the REPORT and TABULATE procedures, you can also create an output data set directly from the procedure.

A simple OUT= in the PROC REPORT statement is all you need to create an output data set. The output data set has one observation for each row in the report. Each summary line is also written as an observation to the output data set as well.

```
      ODS LISTING;
proc report data=furniture nowd out=totaltables;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
run;
ODS HTML BODY='totalreport.html' STYLE=sasweb;
proc print data=totaltables; run;
      ODS HTML CLOSE;
```

Figure 19.13 shows the PROC PRINT HTML report that displays the output data set TOTALTABLES. You can see how closely the structure of the data set TOTALTABLES matches the table layout that we've seen in previous examples. PROC REPORT adds the _BREAK_ column to the output data set, and this column is not present in the tables that we saw in previous examples. The _BREAK_ variable contains the value of the break variable for the summary row observations and a blank for detail row observations. In the preceding PROC REPORT step, the values of _BREAK_ are blank, because there are not any BREAK statements or COMPUTE BEFORE/AFTER blocks requested.

The data set created by the following PROC REPORT step shows how PROC REPORT fills the _BREAK_ column when subtotals and totals are added with BREAK and RBREAK statements. Figure 19.14 shows the PROC PRINT HTML report on the data set TOTALTABLES2 .

```
      ODS LISTING;
proc report data=furniture nowd out=totaltables2;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
  break after Type / summarize;
  rbreak after / summarize;
run;
ODS HTML BODY='totalreport2.html' STYLE=sasweb;
proc print data=totaltables2; run;
      ODS HTML CLOSE;
```

Notice that the data set has grown to include rows for the subtotals and totals. For each of the new rows, the _BREAK_ column shows what type of total it is. The three subtotals are labeled "Type," which is the name of the variable being subtotaled. The main total is labeled "_RBREAK_" to indicate that it is the report total.

Before SAS 9.2, in order to create an output data set, you must have an output destination open. This example turns on the LISTING destination. Be sure to add an ODS LISTING statement before the PROC REPORT step if you have previously turned off listing output.

Figure 19.13

Obs	Type	Material	Price	_BREAK_
1	Coffee	Glass/Metal	100.82	
2	Coffee	Wood	$60.94	
3	Dining	Glass/Metal	400.48	
4	Dining	Wood	195.68	
5	End	Glass/Metal	103.89	
6	End	Wood	$59.05	

Figure 19.14

Obs	Type	Material	Price	_BREAK_
1	Coffee	Glass/Metal	100.82	
2	Coffee	Wood	$60.94	
3	Coffee		$78.91	Type
4	Dining	Glass/Metal	400.48	
5	Dining	Wood	195.68	
6	Dining		275.00	Type
7	End	Glass/Metal	103.89	
8	End	Wood	$59.05	
9	End		$76.76	Type
10			175.19	_RBREAK_

Controlling Column Widths with PROC REPORT

For listing output, you can use the COLWIDTH= option in the PROC REPORT statement or the WIDTH= option in the DEFINE statement to control column widths. For the other destinations, these options have no effect. Instead, the CELLWIDTH= or WIDTH= style attribute allows you to control the width of columns in your PROC REPORT output. You can specify the attribute several different ways: in the PROC REPORT statement, which sets a default width for all columns; in a COMPUTE statement, which sets the default width for LINE output; or in a DEFINE statement, which sets the default width for the column or columns listed in the DEFINE statement.

You can specify the dimension value for CELLWIDTH= in different units. For example, you can specify a percentage (which might be useful if you wanted to specify the width of the whole table). Or you can specify the value in centimeters (CM), inches (IN), millimeters (MM), points (PT), or even EM or EN typesetting units. This example and those that follow specify the dimension in inches.

Without any CELLWIDTH= attribute specified, the output is sized as needed to accommodate the total width of the report. Figure 19.15, which was produced by the following PROC REPORT step, shows the column widths that the procedure and output destination determine.

```
ODS HTML BODY='stdcolumns.html' STYLE=sasweb;
proc report data=furniture nowd;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
  break after Type / summarize;
  rbreak after / summarize;
  compute before _page_ / LEFT;
    LINE "Totally Tables, Inc.";
  endcomp;
run;
ODS HTML CLOSE;
```

The following COMPUTE block changes the width of the header cell above the table to seven inches. Figure 19.16 shows the report if you replace the COMPUTE block in the

preceding program with this new COMPUTE block. Note how all the column widths have been adjusted evenly to fit beneath the header cell.

```
compute before _page_ / left style(Header)=[cellwidth=7in];
```

The preceding example shows how to change the width of the entire table header. The following example uses DEFINE statements to use a separate CELLWIDTH= attribute for each column. Each DEFINE statement specifies a different width. Figure 19.17 shows the results if these DEFINE statements replace the DEFINE statements in the preceding PROC REPORT step and if the CELLWIDTH=7in attribute is removed from the COMPUTE statement.

```
define Type / group 'Table Type' style(Column)=[cellwidth=.5in];
define Material / group 'Construction'
    style(Column)=[cellwidth=1in vjust=b];
define Price / analysis mean 'Average Price' format=dollar8.
    style(Column)=[cellwidth=.75in];
```

Figure 19.15

Totally Tables, Inc.		
Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Coffee		*$79*

Figure 19.16

Totally Tables, Inc.		
Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Coffee		*$79*

Figure 19.17

Totally Tables, Inc.		
Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Coffee		*$79*

The WIDTH= alias for CELLWIDTH= was added in SAS 9.2, so for SAS 9.1.3 you must use CELLWIDTH=.

Inserting Blank Lines with PROC REPORT

When sending output to the LISTING destination, you can add the SKIP option to the BREAK statement in PROC REPORT to insert a blank line between summary groups. The SKIP option, however, can be used only with the LISTING destination.

If you want to send your output to a destination other than LISTING, you need to use COMPUTE blocks and LINE statements to insert blank lines in your report. In a COMPUTE block, you can use the LINE statement to write out a blank line between groups or to add extra blank lines to report headers or footers. For example, you might want the report header "Totally Tables, Inc." to be preceded by a blank line. Or, you might want to put a blank line between summary groups in the report.

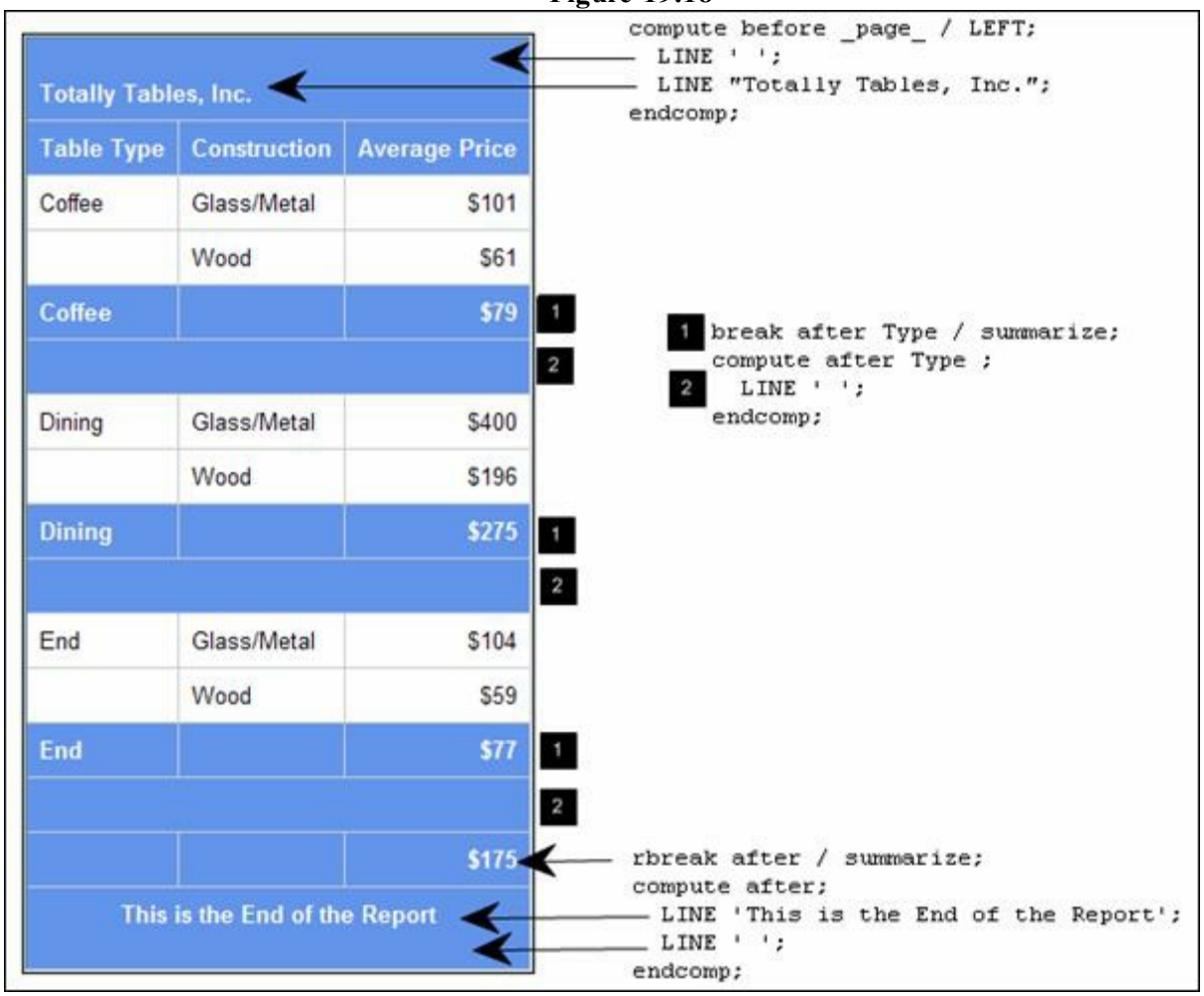
The following PROC REPORT step inserts a blank line before the title in the header cell, after the summary row for each value of TYPE, and after the last sentence at the bottom of the report. In addition, the program specifies the STYLE= option in the PROC REPORT statement for the summary and lines sections of the report. Both STYLE= options specify the Header style element, which ensures that the BREAK-related lines, from the BREAK statement and from the LINE statements, are formatted with the Header style from the Sasweb template.

```
ODS HTML BODY='blanks.html' STYLE=sasweb;
proc report data=furniture nowd
        style(summary)=Header style(lines)=Header;
  column Type Material Price;
  define Type / group 'Table Type';
  define Material / group 'Construction';
  define Price / analysis mean 'Average Price' format=dollar8.;
  compute before _page_ / LEFT;
    LINE ' ';
    LINE "Totally Tables, Inc.";
  endcomp;
  break after Type / summarize;
  compute after Type;
      LINE ' ';
  endcomp;
  rbreak after / summarize;
```

```
  compute after;
    LINE 'This is the End of the Report';
        LINE ' ';
  endcomp;
run;
ODS HTML CLOSE;
```

Figure 19.18 has been annotated so that you can see the statements that produced each summary line and blank line in the header, footer, and body of the report. Notice how the line in the header at the top of the report is left-justified because of the LEFT attribute, whereas the line in the footer of the report uses the default center justification.

Figure 19.18



Creating Cleaner PROC REPORT Output Using SPANROWS

Before ODS, PROC REPORT didn't have table cells with rows and borders unless you specified the BOX option. With ODS, most styles put borders on table cells. When you have GROUP or ORDER variables in your PROC REPORT, the table can end up looking messy because you can see the borders on the cells that are rendered blank by the GROUP or order statement.

For example, the following code produces a table where there are three blank cells in the "Table Type" column:

```
ODS HTML BODY='no_span.html' style=sasweb;
proc report data=furniture nowd;
   column Type Material Price;
   define Type / group 'Table Type';
   define Material / group 'Construction';
   define Price / analysis mean 'Average Price' format=dollar8.;
run;
ODS HTML CLOSE;
```

For the LISTING destination, shown in Figure 19.19, there are no borders, so you just see a blank cell. But for the HTML destination, shown in Figure 19.20, you see the borders on the three empty cells.

By using the SPANROWS option in the PROC REPORT statement, you can remove these extra borders. Here is the modified code:

```
ODS HTML BODY='span.html' style=sasweb;
proc report data=furniture spanrows nowd;
   column Type Material Price;
   define Type / group 'Table Type';
   define Material / group 'Construction';
   define Price / analysis mean 'Average Price' format=dollar8.;
run;
ODS HTML CLOSE;
```

The revised table is shown in Figure 19.21. The three blank cells are now merged into the cell above, and there are no extra lines. This is a simple example. The difference would be more pronounced in a more complex table. In general, for ODS output, SPANROWS is a good option to use on every PROC REPORT.

Figure 19.19

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.20

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.21

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Changing PROC TABULATE Row and Column Heading Style Attributes

Applying style attributes to PROC TABULATE output is similar to applying them in PROC REPORT. All you have to do is add STYLE= options to whichever part of the output you want to change.

There are a number of ways you can use STYLE= in PROC TABULATE. To keep things simple, we look at just one technique: adding STYLE= options to the CLASS and VAR statements. We start with a table similar to the previous examples.

```
      ODS HTML BODY='tabstock.html' STYLE=sasweb;
proc tabulate data=furniture;
  class Type Material;
  var Price;
  table Type='Table Type'*
      Material='Construction',
      Price='Average Price'*Mean=" "*f=dollar8.;
run;
ODS HTML CLOSE;
```

This code produces the output shown in Figure 19.22. It's a basic table using the Sasweb style template. The typeface used in the headings is Arial. As an example, we will change this typeface to Times New Roman.

We do this by adding a STYLE= option to the CLASS and VAR statements for each heading you want to change. Following the STYLE= option, you list the attribute you want to change. A list of the attributes you can modify is shown in "Style Attributes for the REPORT, TABULATE, and PRINT Procedures" at the end of this chapter.

The attribute must be enclosed in square brackets [ ] or curly brackets { }. The following code changes the font typeface to Times New Roman. Notice the use of quotation marks around the typeface name. Because "Times New Roman" contains spaces, these quotation marks are necessary.

```
      ODS HTML BODY='tabstock2.html' STYLE=sasweb;
proc tabulate data=furniture;
  class Type Material / STYLE=[FONT_FACE="Times New Roman"];
  var Price / STYLE=[FONT_FACE="Times New Roman"];
  table Type='Table Type'*
      Material='Construction',
      Price='Average Price'*Mean=" "*f=dollar8.;
run;
ODS HTML CLOSE;
```

The results are shown in Figure 19.23. You can use this technique to change any of the row or column header attributes. You can also use different style attributes for each classification variable by creating two CLASS statements: one for TYPE and one for MATERIAL. Then you could use a different STYLE= option for each variable. The same technique also works when you have two analysis variables: you can use two VAR statements with two different STYLE= options.

Figure 19.22

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.23

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Changing PROC TABULATE BOX= Option Formatting

A common technique in PROC TABULATE is to use the BOX= option in the TABLE statement to put a title in the box at the top left corner of the table. The following code shows how the option is used:

```
      ODS HTML BODY='tabbox.html' STYLE=sasweb;
proc tabulate data=furniture;
  class Type Material;
  var Price;
  table Type='Table Type'*
      Material='Construction',
      Price='Average Price'*Mean=" "*f=dollar8.
      / Box='Retail Stock Report';
run;
ODS HTML CLOSE;
```

The resulting table is shown in Figure 19.24. Notice how the box label is centered in the box. The table might look better if the label were aligned with the left side of the table. That way it would match the row headings. To do this, you can apply a STYLE= option to the BOX= option.

First, you have to specify the box label ("Retail Stock Report") slightly differently from the preceding code. Follow the BOX= option with the LABEL= and STYLE= options enclosed in brackets. Assign the box text to the LABEL= option. Specify style changes with the STYLE= option. The style option changes are enclosed in another set of brackets.

```
      ODS HTML BODY='tabbox2.html' STYLE=sasweb;
proc tabulate data=furniture;
  class Type Material;
  var Price;
  table Type='Table Type'*
      Material='Construction',
      Price='Average Price'*Mean=" "*f=dollar8.
      / Box=[LABEL='Retail Stock Report' STYLE=[JUST=L]];
run;
ODS HTML CLOSE;
```

The revised table is shown in Figure 19.25.

Figure 19.24

Retail Stock Report		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.25

Retail Stock Report		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Changing PROC TABULATE Table Cell Style Attributes

In addition to being able to control style attributes for PROC TABULATE column headings, you can also control style attributes for the cells under a particular heading. This example shows how to change the background color for the table cells with the dollar amounts.

You can do this by adding a STYLE= option to the PROC TABULATE statement. The following code changes the background color to a shade of gray (cxDDDDDD). The BACKGROUND= keyword in the PROC TABULATE statement affects the analysis cells, not the heading cells.

```
      ODS HTML BODY='tabback.html' STYLE=sasweb;
proc tabulate data=furniture STYLE=[BACKGROUND=cxDDDDDD];
  class Type Material;
  var Price;
  table Type='Table Type'*
      Material='Construction',
      Price='Average Price'*Mean=" "*f=dollar8.;
run;
ODS HTML CLOSE;
```

This code produces the output shown in Figure 19.26. You can see that the table cells with the dollar amounts now have a gray background. Another thing we can change is the font weight. The data values are fairly faint against the gray background. We can fix this by adding a second style attribute to the STYLE= option. The second attribute changes the font to bold.

```
      ODS HTML BODY='tabback2.html' STYLE=sasweb;
proc tabulate data=furniture
          STYLE=[BACKGROUND=cxDDDDDD FONT_WEIGHT=bold];
  class Type Material;
  var Price;
  table Type='Table Type'*
      Material='Construction',
      Price='Average Price'*Mean=" "*f=dollar8.;
run;
ODS HTML CLOSE;
```

The results are shown in Figure 19.27. This technique can be used to change many other attributes of PROC TABULATE table cells.

Figure 19.26

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.27

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Changing PROC TABULATE Row Style Attributes

To change the style attributes of a row heading in a PROC TABULATE table, you can use a STYLE= option in the CLASS or VAR statements that build the heading. To change the style attributes of an entire row, you use a different technique. This example shows how to create rows of alternating colors.

The first thing you need to do is set up a format that contains the colors you want to use for each value. In this case, we want the first and third row to be one shade of gray and the third row to be a different shade of gray. The following PROC FORMAT code creates a SAS format that can assign a color to each value of the variable TYPE:

```
    proc format;
  value $typecol 'Dining'='cxBBBBBB'
                 other='cxDDDDDD';   ;
run;
```

Next, this format is used to define the BACKGROUND= value for the variable TYPE. This is done by adding a CLASSLEV statement to our code. The CLASSLEV statement enables you to apply a STYLE= option to individual class variables.

```
    ODS HTML BODY='tabrows.html' STYLE=sasweb;
proc tabulate data=furniture;
  class Type Material;
  classlev Type / STYLE=[BACKGROUND=$typecol.];
  var Price;
  table Type='Table Type'*
      Material='Construction',
      Price='Average Price'*Mean=" "*f=dollar8.;
run;
ODS HTML CLOSE;
```

The resulting table is shown in Figure 19.28. This output has alternating colors for the row headers. Now we can work on continuing that color scheme all of the way across the rows. We need to apply a STYLE= option to the row headings for MATERIAL and also to the data cells. To fix the row headings for MATERIAL, add another CLASSLEV statement for MATERIAL, and set its STYLE= option to <parent>. This setting means that these cells inherit their formatting from the parent cells— the first heading in the row.

```
classlev Material / STYLE=<parent>;
```

To fix the table cells, you need to add a STYLE= option to the TABLE statement. To change the row style attributes, add this option to the row definition. This STYLE= option is also assigned to <parent>.

```
 table Type='Table Type'*
Material='Construction'*[STYLE=<parent>],
Price='Average Price'*Mean=" "*f=dollar8.;
```

The new results are shown in Figure 19.29. Now the entire row inherits its formatting from the style attributes assigned to the variable TYPE. You can use this parent-child relationship to build complex row designs.

Figure 19.28

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.29

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

This technique works best if you have a row variable with a small number of values. Otherwise, you have to do a lot of typing to create the appropriate format. In this case, with only three values, it was easy to set up the PROC FORMAT code.

Changing PROC TABULATE Subtotals and Totals Style Attributes

A common technique in PROC TABULATE is to create totals at the end of each group of results. PROC TABULATE gives you some basic control over these totals. You can change their labels and format their values. However, if you want to do fancier formatting of your subtotals and totals, you will want to turn to ODS. Subtotals and totals are created using the ALL keyword, as shown in the following code. This table uses ALL twice: the first reference computes subtotals and the second computes totals.

```
      ODS HTML BODY='totals.html' STYLE=sasweb;
proc tabulate data=furniture;
  class Type Material;
  var Price;
  table Type='Table Type'*
      (Material='Construction' ALL)
      ALL,
      Price='Average Price'*Mean=" "*f=dollar8.;
run;
ODS HTML CLOSE;
```

The resulting table is shown in Figure 19.30. This table has all of the results we need, but it could be made easier to read. By adding some formatting to the subtotals and totals, we can make them stand out. The STYLE= option for the subtotals is used to create bold type. A second STYLE= option makes the same change for the totals. The style attributes are applied in the TABLE statement, and an asterisk links the attributes to the desired part of the table.

```
      ODS HTML BODY='totals2.html' STYLE=sasweb;
proc tabulate data=furniture;
  class Type Material;
  var Price;
  table Type='Table Type'*
      (Material='Construction' ALL*[STYLE=[FONT_WEIGHT=bold]])
      ALL*[STYLE=[FONT_WEIGHT=bold]],
      Price='Average Price'*Mean=" "*f=dollar8.;
run;
ODS HTML CLOSE;
```

The revised table is shown in Figure 19.31.

Figure 19.30

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
	All	$79
Dining	Construction	
	Glass/Metal	$400
	Wood	$196
	All	$275
End	Construction	
	Glass/Metal	$104
	Wood	$59
	All	$77
All		$175

Figure 19.3

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
	All	**$79**
Dining	Construction	
	Glass/Metal	$400
	Wood	$196
	All	**$275**
End	Construction	
	Glass/Metal	$104
	Wood	$59
	All	**$77**
All		**$175**

Adding Traffic Lighting to PROC TABULATE Table Values

Just as in the earlier example for PROC REPORT, colors can be applied to different ranges of PROC TABULATE table values. If you're creating a large table, this technique is great for focusing the reader's attention on the key results. You use red for bad results, yellow for neutral results, and green for good results. In this example, we take our familiar table and use traffic lighting to mark low prices in green, high prices in red, and prices in between in yellow.

The first thing we need to do is create a format that assigns the value ranges for PRICE to the desired colors. This is very different from the previously discussed PROC REPORT approach. The three color settings are RGB values for green, yellow, and red. The shades of these colors were chosen to be a bit darker than a true green, yellow, or red so that they would stand out against the white table background.

```
    proc format;
  value traffic low-100='cx006600'
              100<-300='cxFF9900'
              300<-high='cxCC0000';
run;
```

To apply these colors, we need to add a STYLE= option to the TABLE statement. Because we want to change the style attributes of the values for PRICE, the STYLE= option is added with an asterisk to the variable PRICE (the option is added to the end of a series of items linked to PRICE). The STYLE= option assigns the FOREGROUND= attribute to the colors given by the TRAFFIC format. To make the colored values easier to read, the type is changed to bold in the same STYLE= option.

```
    ODS HTML BODY='tabtraffic.html' STYLE=sasweb;
proc tabulate data=furniture;
  class Type Material;
  var Price;
  table Type='Table Type'*
      Material='Construction',
      Price='Average Price'*Mean=" "*f=dollar8.*
      [STYLE=[FOREGROUND=traffic. FONT_WEIGHT=bold]];
run;
ODS HTML CLOSE;
```

The results are shown in Figure 19.32. The prices are all colored to draw attention to the high and low values. You can also use this technique in reverse by coloring the backgrounds of the cells in green, yellow, and red instead of the text. To do this, remove the FOREGROUND= attribute and replace it with the BACKGROUND= attribute. It is recommended that you pick slightly lighter shades of the three colors so they don't overwhelm the foreground text. Suggested colors are cx00CC33 for green, cxFFFF66 for yellow, and cxFF3300 for red. Keep the fonts in bold so that the text stands out. This version of the table is illustrated in Figure 19.33.

You can also use this technique with PROC PRINT and PROC REPORT.

Figure 19.32

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.33

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

If you are going to use this technique, you should probably also take the time to design a style definition with a color scheme that is a better match for the red, yellow, and green traffic lighting. A simpler black and white color scheme might be the best approach.

Adding Graphics to PROC TABULATE Tables

We have changed fonts, colors, and other formatting aspects of PROC TABULATE output. However, so far we've just been working with the basic output. This example shows how to add something new to the output: a graphic.

In this example, we are going to place a graphic image in the table box. A graphic can be added in the same way you modified the style attributes for the box in a previous example. The style attribute information is added with a STYLE= suboption on the BOX= option. The PREIMAGE= style attribute enables you to name an image that precedes the text in the cell.

```
      ODS HTML BODY='tabpic.html' STYLE=sasweb;
proc tabulate data=furniture;
  class Type Material;
  var Price;
  table Type='Table Type'*
      Material='Construction',
      Price='Average Price'*Mean=" "*f=dollar8.
      / BOX=[LABEL='Totally Tables, Inc.'
            STYLE=[PREIMAGE='dining.gif']];
run;
ODS HTML CLOSE;
```

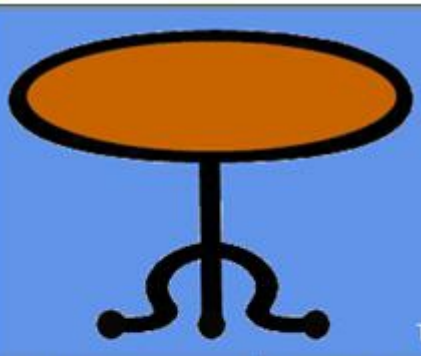
This code creates the output shown in Figure 19.34. The graphic has been added just before the label for the table box. However, the table looks a bit odd now, because the box label does not line up with the column label "Average Price."

To fix this, we can change the justification for the PRICE column. Because the box label is aligned with the bottom of the row, you can change the column label to do the same. This is done with a STYLE= option in the VAR statement for the variable PRICE. Setting the vertical justification to bottom (VJUST=B) should solve the problem.

```
      ODS HTML BODY='tabpic2.html' STYLE=sasweb;
proc tabulate data=furniture;
   class Type Material;
   var Price / STYLE=[VJUST=B];
   table Type='Table Type'*
       Material='Construction',
       Price='Average Price'*Mean=" "*f=dollar8.
       / BOX=[LABEL='Totally Tables, Inc.'
             STYLE=[PREIMAGE='dining.gif']];
run;
ODS HTML CLOSE;
```

The revised table is shown in Figure 19.35. Now the headings line up.

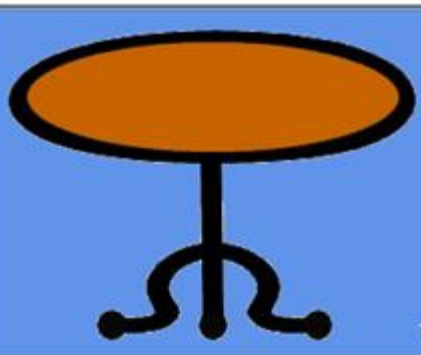
Figure 19.34



Totally Tables, Inc.

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Figure 19.35



Totally Tables, Inc.

Table Type	Construction	Average Price
Coffee	Glass/Metal	$101
	Wood	$61
Dining	Glass/Metal	$400
	Wood	$196
End	Glass/Metal	$104
	Wood	$59

Creating an Output Data Set from PROC TABULATE

With other procedures, you use ODS OUTPUT to create an output data set. With the REPORT and TABULATE procedures, you can also create an output data set directly from the procedure. This is much easier than using ODS, because you don't have to worry about output object names or multiple output data set names.

A simple OUT= option in the PROC TABULATE statement is all you need to tell SAS to generate a data set with the same information as the tabular output.

```
      proc tabulate data=furniture OUT=totaltables;
  class Type Material;
  var Price;
  table Type='Table Type'*
      Material='Construction',
      Price='Average Price'*Mean=" "*f=dollar8.;
run;
ODS HTML BODY='tabtotals.html' STYLE=sasweb;
proc print data=totaltables; run;
ODS HTML CLOSE;
```

Figure 19.36 shows the PROC PRINT HTML report of the output data set TOTALTABLES. The data set closely matches the tabular output. PROC TABULATE adds three columns that show where each row of data came from in the output data set. These columns are _TYPE_, _PAGE_, and _TABLE_ . In this example, _TABLE_ and _PAGE_ do not come into play because we have only one table and only one page. The value of _TYPE_ indicates which combination of CLASS variables is in each row.

The data set created by the following PROC TABULATE step shows how the _TYPE_ column changes when computation of subtotals and totals is added to the program. Figure 19.37 shows the PROC PRINT HTML report on data set TOTALTABLES2 .

```
      proc tabulate data=furniture OUT=totaltables2;
  class Type Material;
  var Price;
  table Type='Table Type'*
      (Material='Construction' ALL) ALL,
      Price='Average Price'*Mean=" "*f=dollar8.;
run;
ODS HTML BODY='tabtotals2.html' STYLE=sasweb;
proc print data=totaltables2; run;
ODS HTML CLOSE;
```

Notice that the data set has grown, with new rows added for the subtotals and totals. Also, the order of the rows is quite different from the order they appear in the table. Unlike the output data set from PROC REPORT, this data set is not simply a copy of the results in the order they appear in the table. Instead, the output data set has one row for every unique combination of the CLASS variables. This is similar to the output data sets created by the MEANS and SUMMARY procedures. If you have a very complex table with lots of nested classifications, the output data set can be very complex and confusing. Use PROC TABULATE output data sets with a good deal of caution and preparation.

Figure 19.36

Obs	Type	Material	_TYPE_	_PAGE_	_TABLE_	Price_Mean
1	Coffee	Glass/Metal	11	1	1	100.816
2	Coffee	Wood	11	1	1	60.943
3	Dining	Glass/Metal	11	1	1	400.483
4	Dining	Wood	11	1	1	195.684
5	End	Glass/Metal	11	1	1	103.892
6	End	Wood	11	1	1	59.052

Figure 19.37

Obs	Type	Material	_TYPE_	_PAGE_	_TABLE_	Price_Mean
1	Coffee	Glass/Metal	11	1	1	100.816
2	Coffee	Wood	11	1	1	60.943
3	Dining	Glass/Metal	11	1	1	400.483
4	Dining	Wood	11	1	1	195.684
5	End	Glass/Metal	11	1	1	103.892
6	End	Wood	11	1	1	59.052
7	Coffee		10	1	1	78.912
8	Dining		10	1	1	275.000
9	End		10	1	1	76.762
10			00	1	1	175.189

Controlling Column Widths with PROC TABULATE

As we saw with PROC REPORT, PROC TABULATE sizes the columns in the table as needed for the destination. In destinations that support the ORIENTATION= option, like RTF and PDF, you can put wide tables onto the page and the column widths are adjusted automatically. The RTS= option in the TABLES statement, which sets the width of the row title, is valid only in the LISTING destination. So, what do you do if you want to explicitly set the column widths of your PROC TABULATE output? As with PROC REPORT, you can use the CELLWIDTH= or WIDTH= style attributes. In PROC TABULATE, you can add these attributes to your CLASS, VAR, and TABLES statements.

The code below demonstrates how ODS defines the default column widths in an HTML report. Figure 19.38 shows the results.

```
ODS HTML FILE='tabwidth.html' STYLE=sasweb;
proc tabulate data=furniture;
  class Type Material;
  var Price;
  table Type='Table Type'*
      (Material='Construction' ALL) ALL,
      Price='Average Price'*Mean=" "*f=dollar8.;
run;
ODS HTML CLOSE;
```

If you add the STYLE= option and either the CELLWIDTH= or WIDTH= attributes to the CLASS statement, you can explicitly define the width of the two classification variables. If you want different widths for the two variables, specify each on its own CLASS statement and set the CELLWIDTH= style attribute on each statement. Replacing the CLASS statement above with the two CLASS statements that follow produces the results in Figure 19.39. The output shows a difference in width of the two classification columns.

```
  class Type / style=[cellwidth=.5in];
  class Material/ style=[width=1in];
```

One CLASS statement uses the CELLWIDTH= attribute and the other CLASS statement uses the WIDTH= attribute, which is an alias for CELLWIDTH=. (CELLWIDTH= is the SAS 8 and SAS®9 attribute name, and WIDTH= is the SAS 9.2 alias for the CELLWIDTH= attribute.)

Alternately, you could control the output width of the entire table proportionally by using the WIDTH= or OUTPUTWIDTH= attribute in the TABLE statement. The following TABLE

statement specifies that the width of the table take up 100% of the page. Replacing the TABLE statement in the PROC step above with the TABLE statement that follows produces the output in Figure 19.40.

```
table Type='Table Type'*
    (Material='Construction' ALL) ALL,
    Price='Average Price'*Mean=" "*f=dollar8. /
    style=[outputwidth=100%];
```

Do not combine these two techniques. Either specify the widths in the TABLE statement or in the CLASS statement(s).

Figure 19.38

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
	All	$79
Dining	Construction	
	Glass/Metal	$400

Figure 19.39

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
	All	$79
Dining	Construction	
	Glass/Metal	$400

Figure 19.40

		Average Price
Table Type	Construction	
Coffee	Glass/Metal	$101
	Wood	$61
	All	$79
Dining	Construction	
	Glass/Metal	$400

Changing PROC PRINT Column Heading Style Attributes

Applying style attributes to PROC PRINT output is very simple. All you have to do is add STYLE= options to whichever part of the output you want to change.

There are a number of ways you can use STYLE= in the PRINT procedure. To keep things simple, we look at just one technique: adding a STYLE= option to the PROC PRINT statement. We start with the following code. It lists three variables.

```
      ODS HTML FILE='listfurniture.html' STYLE=sasweb;
proc print data=furniture(obs=5) noobs label;
  var Type Material Price;
run;
ODS HTML CLOSE;
```

This code produces the output shown in Figure 19.41. It's a basic table formatted with the Sasweb template. The typeface used in the headings and table body is Arial. As an example, we will change this typeface to Times New Roman.

We do this by adding a STYLE= option to the PROC PRINT statement. The STYLE keyword is followed by the name of the style element that you want to modify. In this case, the element is called Header. A list of the elements you can modify is in the SAS OnlineDoc for PROC PRINT.

Following the name of the style element, use an equal sign, list the attributes you want to change, and enclose the list in brackets. Details about these attributes and their settings can be found in the SAS OnlineDoc for ODS.

The attribute must be enclosed in square brackets [ ] or curly brackets { }. The following code changes the font typeface to Times New Roman. Notice the use of quotation marks around the typeface name. Because "Times New Roman" contains spaces, these quotation marks are necessary.

```
      ODS HTML BODY='listfurniture2.html' STYLE=sasweb;
proc print data=furniture(obs=5) noobs label
                STYLE(Header)=[FONT_FACE='Times New Roman'];
  var Type Material Price;
run;
ODS HTML CLOSE;
```

The results are shown in Figure 19.42. This technique can be used to change any of the column header attributes.

Figure 19.41

Retail Stock Report

Type of Table	Table Material	Table Price
Coffee	Wood	$68.53
Coffee	Wood	$49.99
Coffee	Glass/Metal	$110.14
Dining	Wood	$346.47
Dining	Wood	$318.54

Figure 19.42

Retail Stock Report

Type of Table	Table Material	Table Price
Coffee	Wood	$68.53
Coffee	Wood	$49.99
Coffee	Glass/Metal	$110.14
Dining	Wood	$346.47
Dining	Wood	$318.54

Changing PROC PRINT Table Cell Style Attributes

In addition to being able to control style attributes for PROC PRINT column headings, you can also control style attributes for the cells under a particular heading. This example shows how to change the background color for the table cells with the dollar amounts.

You can do this by adding a STYLE= option to the VAR statement. However, because you only want to apply the style attributes to the variable PRICE, you need to create two VAR statements. The first VAR statement specifies the first two variables. The style attributes for these variables will not be changed. The second VAR statement specifies only the variable PRICE. At the end of the second VAR statement, add a slash (/) and then follow it with a STYLE= option.

The following code changes the background color to a shade of gray (cxDDDDDD). Notice that this time we are not using the name of a style element. In the last example, we used STYLE(Header) to specify that the new style attributes be applied to the table headers. This time, because we are applying the style attributes in the definition for PRICE, they are automatically applied to that column. There is no need to specify a style element.

```
      ODS HTML BODY='listfurniture3.html' STYLE=sasweb;
proc print data=furniture(obs=5) noobs label;
  var Type Material;
  var Price / STYLE=[BACKGROUND=cxDDDDDD];
run;
ODS HTML CLOSE;
```

This code produces the output shown in Figure 19.43. You can see that the table cells with the dollar amounts now have a gray background. The column heading also has a gray background. When you apply a style attribute in the VAR statement, it applies to the entire column for that variable.

If you want only the column values to have the gray background, modify your STYLE= option setting to specify the location of where it applies. The PRINT procedure has a style element called Data, which controls the appearance of the data in each column, and this element does not affect the header.

```
      ODS HTML BODY='listfurniture4.html' STYLE=sasweb;
proc print data=furniture(obs=5) noobs label;
  var Type Material;
  var Price / STYLE(Data)=[BACKGROUND=cxDDDDDD];
run;
ODS HTML CLOSE;
```

The results are shown in Figure 19.44. This technique can be used to change the color of any of the table cells.

Figure 19.43

Retail Stock Report

Type of Table	Table Material	Table Price
Coffee	Wood	$68.53
Coffee	Wood	$49.99
Coffee	Glass/Metal	$110.14
Dining	Wood	$346.47
Dining	Wood	$318.54

Figure 19.44

Retail Stock Report

Type of Table	Table Material	Table Price
Coffee	Wood	$68.53
Coffee	Wood	$49.99
Coffee	Glass/Metal	$110.14
Dining	Wood	$346.47
Dining	Wood	$318.54

Changing PROC PRINT Style Attributes for Totals

A common technique in PROC PRINT is to create totals at the end of your of results. PROC PRINT creates totals for the variable you list in the SUM statement, as shown in the following code.

```
      ODS HTML BODY='prttotals.html' STYLE=sasweb;
proc print data=furniture noobs label;
  var Type Material Price;
  sum Price;
run;
ODS HTML CLOSE;
```

The resulting table is shown in Figure 19.45. This table has all of the results we need, but it could be made easier to read. By adding some formatting to the total, you can make the total stand out. In the following PROC PRINT step, a STYLE= option in the SUM statement applied to the "GrandTotal" area adds some space before the total by increasing the cell height, vertically centers the text in the cell, and changes the font color to black.

```
      ODS HTML BODY='prttotals2.html' STYLE=sasweb;
proc print data=furniture noobs label;
  var Type Material Price;
  sum Price / Style(GrandTotal)=[VJUST=C CELLHEIGHT=50 COLOR=BLACK];
run;
ODS HTML CLOSE;
```

The bottom of the revised table is shown in Figure 19.46. This technique can also be used to modify subtotals created by the SUMBY statement. The style elements in that case are Total (for the value) and ByLabel (for the label).

Figure 19.45

Dining	Wood	$109.45
Dining	Glass/Metal	$256.18
Dining	Glass/Metal	$238.11
Coffee	Glass/Metal	$132.80
End	Wood	$81.18
Dining	Glass/Metal	$422.39
Dining	Wood	$53.97
		$357,910.56

Figure 19.46

Dining	Wood	$109.45
Dining	Glass/Metal	$256.18
Dining	Glass/Metal	$238.11
Coffee	Glass/Metal	$132.80
End	Wood	$81.18
Dining	Glass/Metal	$422.39
Dining	Wood	$53.97
		$357,910.56

Indenting Values with Margin Attributes Using LEFTMARGIN=

Before SAS 9.2, the only way to indent variable values was to pad the front of the variable value with non-breaking spaces (either using the HTML entity   or using the ODS ESCAPECHAR plus an underscore non-breaking space control sequence). With SAS 9.2, the new attribute LEFTMARGIN= (or alias MARGINLEFT=) can be added to your STYLE= options to indent values.

The LEFTMARGIN= attribute tells the ODS destination where the information in a cell should start relative to the left-side cell wall. This attribute was designed to work directly in procedure syntax with ODS RTF and ODS PDF. For the ODS HTML destination, however, you still need to pass the style attribute into the ODS HTML destination using the ODS ESCAPECHAR style override method.

The following PROC REPORT DEFINE statement shows how you can code LEFTMARGIN=. It is used for the first column, labeled “Furniture Table Type.”

```
define type / group 'Furniture Table Type'
    style(column)={leftmargin=.25in};
```

The results of this code are shown in Figure 19.47 for RTF and PDF output.

There are two downsides to this technique: it doesn’t work with PROC TABULATE, and it doesn’t work with ODS HTML (as seen in the first column of Figure 19.48). If you use this direct technique with PROC TABULATE, as shown here:

```
classlev type material / style={leftmargin=.25in};
```

SAS generates this warning message in the SAS log for the PROC TABULATE step:

```
559  proc tabulate data=furniture f=10.2;
560    title 'This syntax does NOT work for TAB in any destination';
561    class type material;
562    classlev type material /
563          style={
563!                leftmargin=.25in};
WARNING: LEFTMARGIN does not make sense in the current context. Attribute will be ignored.
564    var price;
565    table type*material,
566         price*(min mean max);
567  run;
```

Although the direct technique works for RTF and PDF destinations, it does not work

for HTML. This means that you either have to use one technique for HTML and a different technique for other destinations, or you can use the method described in the next example, which works in all three destinations. As you can see by comparing Figures 19.48 and 19.51 this next technique, used on the column labeled "Type With Format," works in all three destinations, and, as an added benefit, it also works with PROC TABULATE.

Figure 19.47: PDF and RTF PROC REPORT Results

PDF Results

Furniture Table Type	Type With Format	Material	Min Price	Avg Price	Max Price
Coffee	Coffee	Glass/Metal	49.99	100.82	174.87
		Wood	49.99	60.94	87.37
Dining	Dining	Glass/Metal	100.16	400.48	698.54
		Wood	50.59	195.68	349.94
End	End	Glass/Metal	49.99	103.89	171.85
		Wood	49.99	59.05	87.38
			49.99	*175.19*	*698.54*

RTF Results

Furniture Table Type	Type With Format	Material	Min Price	Avg Price	Max Price
Coffee	Coffee	Glass/Metal	49.99	100.82	174.87
		Wood	49.99	60.94	87.37
Dining	Dining	Glass/Metal	100.16	400.48	698.54
		Wood	50.59	195.68	349.94
End	End	Glass/Metal	49.99	103.89	171.85
		Wood	49.99	59.05	87.38
			49.99	*175.19*	*698.54*

Figure 19.48: HTML PROC REPORT Results

Furniture Table Type	Type With Format	Material	Min Price	Avg Price	Max Price
Coffee	Coffee	Glass/Metal	49.99	100.82	174.87
		Wood	49.99	60.94	87.37
Dining	Dining	Glass/Metal	100.16	400.48	698.54
		Wood	50.59	195.68	349.94
End	End	Glass/Metal	49.99	103.89	171.85
		Wood	49.99	59.05	87.38
			49.99	*175.19*	*698.54*

Indenting Values with Margin Attributes

You need to use a different technique with PROC TABULATE, and the technique that works for PROC TABULATE also works for ODS HTML. The method involves using ODS ESCAPECHAR techniques to alter style attributes.

The following program demonstrates how to use the escape character technique and the direct technique. It starts with a PROC FORMAT step that defines two formats whose value labels include the LEFTMARGIN= attribute. Both formats specify value labels that start with the caret (^) escape character, followed by the STYLE keyword and style attributes. The entire STYLE specification is enclosed in curly brackets, and each style attribute specification is enclosed in square brackets. This entire escape sequence precedes the value label text.

The first TYPE column in the following PROC REPORT code specifies LEFTMARGIN= explicitly in the DEFINE statement, while the second TYPE column (alias FMTTYP) and the MATERIAL column apply formats to insert left margins. The PROC TABULATE code inserts left margins by applying formats to TYPE and MATERIAL. The PDF, RTF, and HTML results are shown in Figure 19.49, Figure 19.50, and Figure 19.51, respectively. Notice in Figure 19.51 that the first occurrence of the TYPE column is not indented in HTML as it is for the other two destinations. However, when you use the user-defined format FMTTYP, as applied to the second column, all three destinations respect the inline specification for LEFTMARGIN=.

```
proc format;
  value $typ 'Coffee' = '^{style[leftmargin=.25in]}Coffee'
             'Dining' = '^{style[leftmargin=.25in]}Dining'
             'End'    = '^{style[leftmargin=.25in]}End';
  value $mat 'Glass/Metal' = '^{style[leftmargin=.25in]}Glass/Metal'
             'Wood'        = '^{style[leftmargin=.25in]}Wood';
run;

ODS PDF FILE='margin_indent.pdf';
ODS RTF FILE='margin_indent.rtf';
ODS HTML BODY='margin_indent.html' STYLE=sasweb;
ODS ESCAPECHAR='^';
proc report data=furniture nowd;
  column type type=fmttyp material price price=avpr price=mxpr;
```

```
  define type / group 'Furniture Table Type'
                    style(column)={leftmargin=.25in};
  define fmttyp / group 'Type With Format' f=$typ.;
  define material / group 'Material' f=$mat.;
  define price / min 'Min Price' f=10.2;
  define avpr / mean 'Avg Price' f=10.2;
  define mxpr / max 'Max Price' f=10.2;
  rbreak after / summarize;
run;
proc tabulate data=furniture f=10.2;
  class type material;
  classlev type material;
  var price;
  table type*material, price*(min mean max);
  format type $typ. material $mat.;
run;
ODS _ALL_ CLOSE;
```

Figure 19.49: PDF Results

Proc REPORT Results

Furniture Table Type	Type With Format	Material	Min Price	Avg Price	Max Price
Coffee	Coffee	Glass/Metal	49.99	100.82	174.87
		Wood	49.99	60.94	87.37
Dining	Dining	Glass/Metal	100.16	400.48	698.54
		Wood	50.59	195.68	349.94

Proc TABULATE Results

		Table Price		
		Min	Mean	Max
Type of Table	Table Material			
Coffee	Glass/Metal	49.99	100.82	174.87
	Wood	49.99	60.94	87.37
Dining	Glass/Metal	100.16	400.48	698.54

Figure 19.50: RTF Results

Proc REPORT Results

Furniture Table Type	Type With Format	Material	Min Price	Avg Price	Max Price
Coffee	Coffee	Glass/Metal	49.99	100.82	174.87
		Wood	49.99	60.94	87.37
Dining	Dining	Glass/Metal	100.16	400.48	698.54
		Wood	50.59	195.68	349.94

Proc TABULATE Results

		Table Price		
		Min	Mean	Max
Type of Table	Table Material			
Coffee	Glass/Metal	49.99	100.82	174.87
	Wood	49.99	60.94	87.37
Dining	Glass/Metal	100.16	400.48	698.54

Figure 19.51: HTML Results

Proc REPORT Results

Furniture Table Type	Type With Format	Material	Min Price	Avg Price	Max Price
Coffee	Coffee	Glass/Metal	49.99	100.82	174.87
		Wood	49.99	60.94	87.37
Dining	Dining	Glass/Metal	100.16	400.48	698.54
		Wood	50.59	195.68	349.94

Proc TABULATE Results

		Table Price		
		Min	Mean	Max
Type of Table	Table Material			
Coffee	Glass/Metal	49.99	100.82	174.87
	Wood	49.99	60.94	87.37
Dining	Glass/Metal	100.16	400.48	698.54

Style Attributes for the REPORT, TABULATE, and PRINT Procedures

In general, the same style attributes are available for use with the STYLE= options for these three procedures. We have touched on only a few of the style attributes available with the REPORT, TABULATE and PRINT procedures. In general, the attributes that you can use with these three procedures are the same as the attributes that you would use in a style template definition on a DEFINE statement.

Most of the attributes that you can use inside these procedures are table or cell attributes. This makes sense, as the only things that you can build with the report procedure syntax are report tables that are composed of header cells and data cells. You cannot change the SystemTitle style attributes or the ProcTitle style attributes inside the procedure syntax; you can do that only in a style template definition.

In SAS 9.2, the ODS style attributes are described in the PROC TEMPLATE documentation under the topic title "Style Attributes and their Values." Specific ODS documentation for PROC REPORT and PROC TABULATE is documented under the topic titles "Using Style Elements in PROC REPORT" and "Using Style Elements in PROC TABULATE," respectively.

The examples in this chapter have shown how to modify individual cells, rows, columns, and headers. More examples of this type are included in the SAS OnlineDoc.

This chapter has not covered how to apply a style attribute to an entire table or report, other than briefly in the last PROC TABULATE example. The syntax is actually quite simple.

For PROC REPORT, use the following syntax:

```
PROC REPORT DATA=dataset STYLE=[style-attribute(s)] NOWD;
```

For PROC PRINT, the syntax is very similar:

```
PROC PRINT DATA=dataset STYLE=[ style-attribute(s)];
```

For PROC TABULATE, the syntax is similar, but the impact is different. A style override in the PROC TABULATE statement applies to all the data cells in the TABULATE output.

```
PROC TABULATE DATA=dataset STYLE=[ style-attribute(s)];
```

To affect the entire table, you need to use a STYLE= option in the TABLE statement:

```
TABLE <<page-definition,> row-definition,> column-definition
/ STYLE=[attribute=value attribute1=value1];
```

This is only a brief overview of the available style attributes. Details about these attributes and their settings can be found in the SAS OnlineDoc for ODS.

Chapter 20
Understanding ODS Graphics Output

So far we've looked only at procedures that produce tabular output. However, you can use both SAS/GRAPH and ODS Statistical Graphics with ODS. This chapter introduces the topic of how to create graphics output using ODS. If you're not familiar with SAS/GRAPH or ODS Statistical Graphics, we hope this introduction will pique your interest.

In SAS 9.2, ODS style templates can be used in the creation of all graphic output. Whether you're using traditional SAS/GRAPH procedures or the new ODS Statistical Graphics, you can create graphs that use colors and fonts that are compatible with your tabular output.

We start the chapter by describing the ODS Statistical Graphics and template-based graphics, new in SAS 9.2. We look at ways to use the ODS GRAPHICS ON/OFF statement with selected SAS procedures and introduce a few examples of the statistical graphics (SG) procedures. We'll also demonstrate how to combine graphs with tabular output.

Additionally, we'll show how to produce graphs that contain drill-down links to static reports using ODS and SAS/GRAPH. You can even use Java applets and ActiveX controls and move into the world of more interactive graphs by using SAS/GRAPH devices designed to produce interactive graphs.

The CANDYBAR and MOVIEPLOTS data sets are used for the examples in this chapter. These data sets are available on the author pages for this book at http://support.sas.com/authors.

Understanding Graphics Creation in SAS 9.2

SAS 9.2 offers you several possibilities for graph creation, some familiar and some completely new. First, traditional device-based SAS/GRAPH procedures (such as GCHART, GPLOT, and GMAP) now use ODS style templates for all ODS destinations, including the LISTING destination. Traditional SAS/GRAPH procedures fall under the category of device-based graphics, because you still use SAS/GRAPH devices and SAS/GRAPH GOPTIONS for image creation.

The new ODS Statistical Graphics framework for graph production uses template-based graphics in several new and exciting ways:

1) Selected SAS procedures support the use of the ODS GRAPHICS ON/OFF statements to produce predefined, template-based statistical graphs.

2) New statistical graphics (SG) procedures, SGPLOT, SGSCATTER and SGPANEL, enable you to create statistical graphs directly from your data.

3) Using the Graph Template Language, you can either modify existing statistical graphs or design your own graphs. Then, you can render the graphs using the SGRENDER procedure.

Both these methods, device-based and template-based, support the use of style template information. However, only template-based graphics use STATGRAPH graph templates. In SAS 9.2, template-based graphics do not use SAS/GRAPH GOPTIONS to alter graph appearance. You either use style and graph template syntax to change colors and fonts for graph style elements, or you can, for the SG procedures, use SG procedure options to alter colors and fonts of particular graph elements. The ODS GRAPHICS statement is the syntax that you use to specify attributes such as image format, image name, etc.

In contrast, device-based graphics use ODS style templates, SAS/GRAPH GOPTIONS and global statements and procedure syntax to affect graph appearance. Of all these methods, style template usage is new with SAS 9.2. The GSTYLE system option controls whether device-based graphics use ODS style templates. The GSTYLE option is turned on by default, but you can turn it off if you want to run legacy code. Here is the syntax for turning the option off and back on:

Turn GSTYLE Off (disables style template use)	Turn GSTYLE On (if it has been turned off)
`OPTIONS NOGSTYLE;` `OPTIONS GSTYLE=OFF;`	`OPTIONS GSTYLE;` `OPTIONS GSTYLE=ON;`

By default, GSTYLE is in effect when you start your SAS session, and the style specified on your ODS statement is used when the graph is created. If you set the NOGSTYLE option, however, your legacy SAS/GRAPH code will work as originally designed. This latter setting might be important if you have device-based SAS/GRAPH code that has been highly customized with GOPTIONS for fonts and colors. The only exception to this behavior is with the client-side devices ActiveX and Java. These devices are called client-side or client

devices because the image is actually rendered by an application on the client machine instead of being created as a fixed image. The Java and ActiveX devices always use style templates to control appearance, and the graph can be manipulated on the client machine because of the way that Java and ActiveX work on the client machine. We talk about ActiveX and Java graphics in upcoming examples.

First, we want to illustrate how GSTYLE works with device-based SAS/GRAPH output. The following example illustrates use of the GSTYLE option with a simple SAS/GRAPH program. Figure 20.1 shows the output with GSTYLE and STYLE=OCEAN. If you use NOGSTYLE and resubmit the same code, your output looks like that shown in Figure 20.2. Note that even though the LISTING destination is specified, SAS uses the OCEAN style template for the graphic output in Figure 20.1 when GSTYLE is in effect. Tabular output sent to the LISTING destination does not use style template information.

```
ODS LISTING STYLE=ocean;

proc gchart data=CandyBars;

  vbar Filling / sumvar=Sales

                 name='out20_1';

run; quit;
```

Figure 20.1
Default GSTYLE in effect

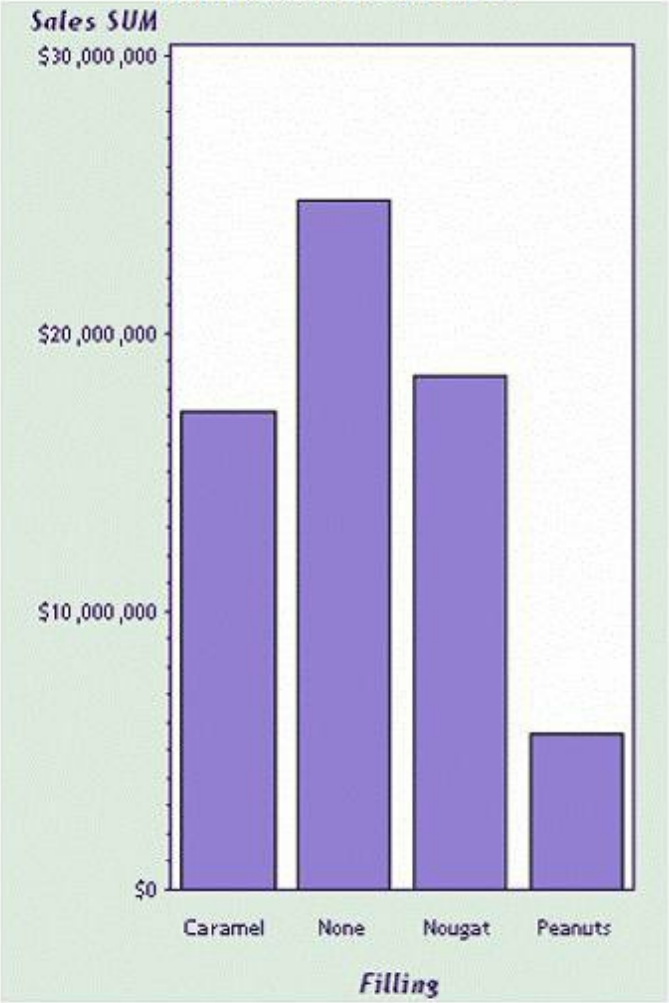
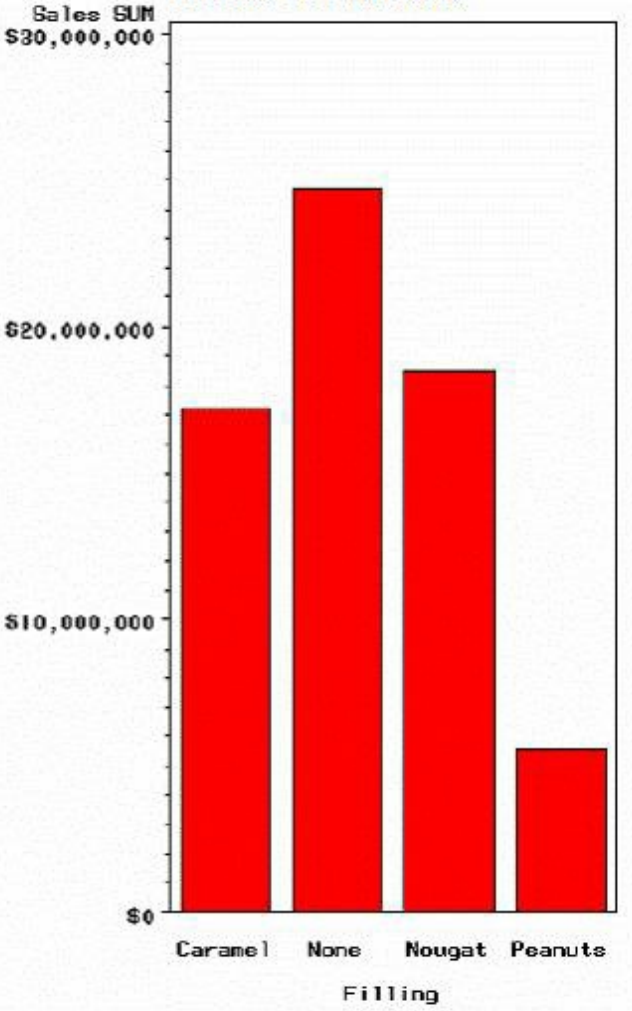


Figure 20.2
NOGSTYLE in effect



If you have highly customized SAS/GRAPH code written in earlier versions of SAS, use the NOGSTYLE system option so your GOPTIONS statements work as expected. In addition, consult the SAS/GRAPH documentation about the use of special Zdevices, which maintain compatibility with earlier device drivers. For example, if your legacy application used the GIF device, you might want to change your code to use the ZGIF device. Consult the documentation topic "Changing the Appearance of Output to Match That of Earlier SAS Releases" for more information.

Generating Graphics Output from SAS Procedures Using the ODS GRAPHICS Statement

Beginning in SAS 9.2, selected SAS procedures were modified to automatically produce statistically relevant images. These procedures include procedures in SAS/STAT, SAS/QC, SAS/HPF, SAS/ETS, SAS Risk Dimensions, and even a few in Base SAS. These procedures use template-based methods to produce images. The documentation topic “Procedures That Support ODS Graphics” lists all the procedures that can generate graphs. Like other ODS templates, you can modify and enhance your procedure’s graph template with PROC TEMPLATE.

Our example uses one Base SAS procedure, PROC UNIVARIATE, and two SAS/STAT procedures, PROC ANOVA and PROC REG. The following program creates an HTML output file. With the LISTING destination open, the program also sends tabular output to the Output window. The ODS SELECT statements limit the output objects sent to each destination. Chapter 8, “Limiting ODS Output,” shows how to select and exclude output objects with the ODS SELECT and ODS EXCLUDE statements. We can do the same with graphic objects.

```
ODS LISTING STYLE=ocean;

ODS HTML PATH='.' (URL=NONE)

                FILE='MovieStats.HTML' STYLE=ANALYSIS;

ODS GRAPHICS ON;

ODS SELECT BasicMeasures PPPlot;

proc univariate data=movieplots;

  var ticketssold;

      ppplot;

run; quit;

ODS SELECT BoxPlot;

proc anova data = movieplots;

  class Theme;

  model ticketssold = ThemeShort;

run; quit;

ODS SELECT FitStatistics FitPlot;

proc reg data = movieplots;
```

```
  model ticketssold = concessions;
run; quit;
ODS GRAPHICS OFF;
ODS _ALL_ CLOSE;
```

Figure 20.3 shows part of the PROC UNIVARIATE tabular output copied from the Output window, and it also shows the PROC UNIVARIATE PPPlot graphic output. The plot is shown in the default picture viewer, which is Windows Photo Gallery.

The ODS HTML output, on the other hand, is contained in two files (HTML and PNG) and the graphic output uses the same color and font information as the tabular output. The use of the PATH= option is the recommended method for creating HTML output that contains graphics output. This method is discussed in the next topic.

The important item to note is that no matter the destination, ODS creates the graphic output using information from SAS style templates. Figure 20.4 shows the tabular and graphical output from PROC UNIVARIATE. Figure 20.5 shows the results of the PROC ANOVA, and Figure 20.6 shows the PROC REG results. Note that some of the results are continued on the same page as Figure 20.7.

Figure 20.3

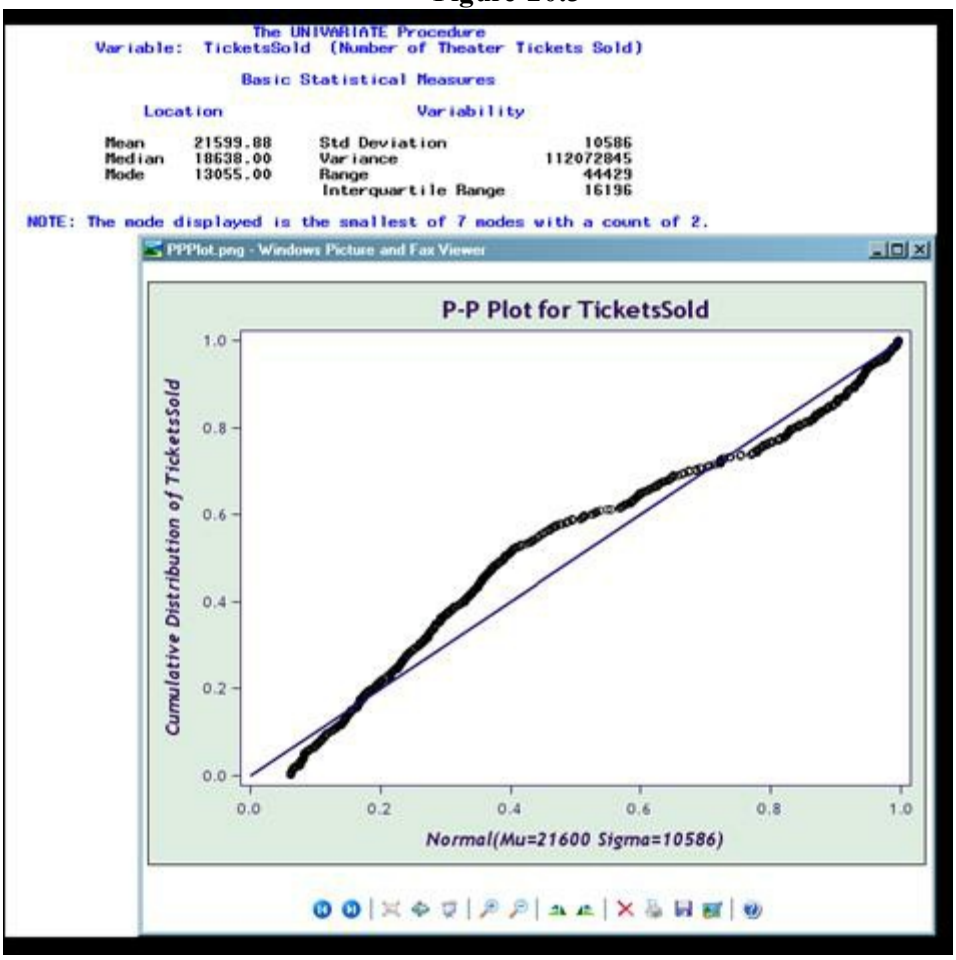


Figure 20.4

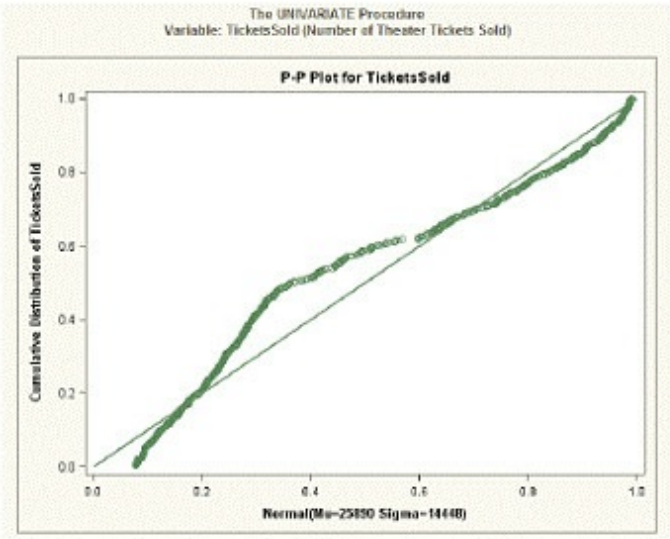
Tabular

The UNIVARIATE Procedure
Variable: TicketsSold (Number of Theater Tickets Sold)

Basic Statistical Measures			
Location		**Variability**	
Mean	25889.82	**Std Deviation**	14448
Median	21077.00	**Variance**	208739566
Mode	15529.00	**Range**	54448
		Interquartile Range	22307

Note: The mode displayed is the smallest of 5 modes with a count of 2.

Graphical



Creating Graphic Images with the ODS Statistical Graphics Framework

Unlike device-based SAS/GRAPH procedures, template-based procedures do *not* create GRSEG entries in a SAS catalog. By default, when you use ODS Statistical Graphics, as in the previous example, ODS creates a PNG (Portable Network Graphics) file for each image and for each output destination.

The previous program created listing output and HTML output. The PPPLOT.PNG image in Figure 20.3 was created when ODS sent the PROC UNIVARIATE plot to the LISTING destination. ODS created a second image file, PPLOT1.PNG, when it sent the same output to the HTML destination.

Additional images are assigned numeric suffixes sequentially during your SAS session. Because both the LISTING and HTML destinations were open for the three procedure steps, the same series of images was created for each destination. Examining the output folder, as excerpted in Figure 20.7, shows that multiple PNG files were created. All of the PNG files with a suffix of "1" were created for the HTML destination, whereas the files without a numeric suffix on the filename were created for the LISTING destination.

The HTML <IMG> tag contains an explicit SRC= value that shows the names used by ODS when the PNG files are created (the tags are edited here to show just the SRC= values):

```
<img src="PPPlot1.png">
<img src="BoxPlot1.png">
<img src="FitPlot1.png">
```

Notice that the previous program did not contain a GSASFILE statement, a GOPTIONS statement, or a DEVICE= GOPTION. The GOPTIONS statement is not needed, since ODS ignores this statement. ODS creates the graphic output and then, based on the destination that you use for the output, it creates the appropriate type of graphics file. For LISTING, LaTeX. and HTML destinations, ODS creates a PNG file by default. For the PDF (Printer family) and RTF destinations, the ODS output graph is embedded in the output document.

You can alter ODS Statistical Graphics defaults in the ODS GRAPHICS ON statement. Selected options are shown in Table 20.1. For a complete list of options and syntax for the ODS GRAPHICS statement, refer to the documentation topic entitled "Using the ODS GRAPHICS Statement."

Table 20.1: Selected ODS GRAPHICS Options

Task	Option
Reset index counter suffix	ods graphics on / reset=index;
Reset all options to their defaults	ods graphics on / reset=all;
Reset multiple options	ods graphics on / reset=height reset=width;
Specify an image type	ods graphics on / imagefmt=gif;
Specify the name of an image	ods graphics on / imagename="TheName";

Specify size information ods graphics on / height=400px width=600px;

Some procedures might have explicit syntax to turn on graph creation, as seen with the PPPLOT statement in PROC UNIVARIATE. PROC ANOVA and PROC REG produce graphs automatically. You need to consult each procedure's documentation to find the appropriate syntax.

Figure 20.5

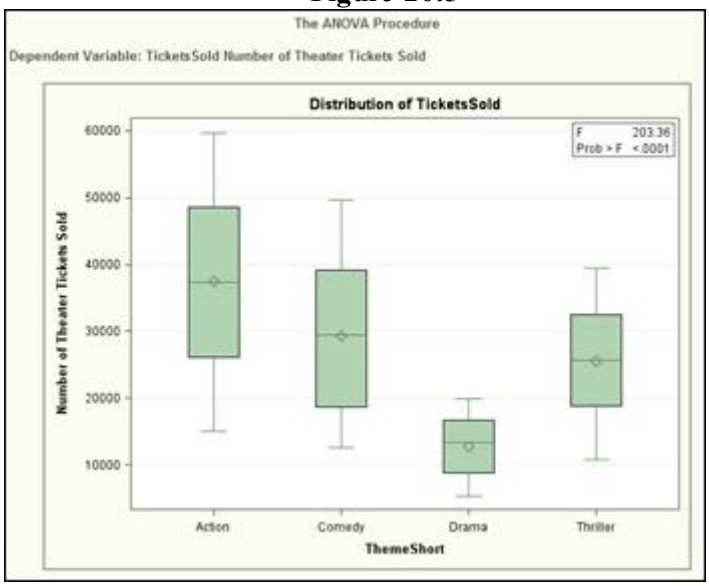
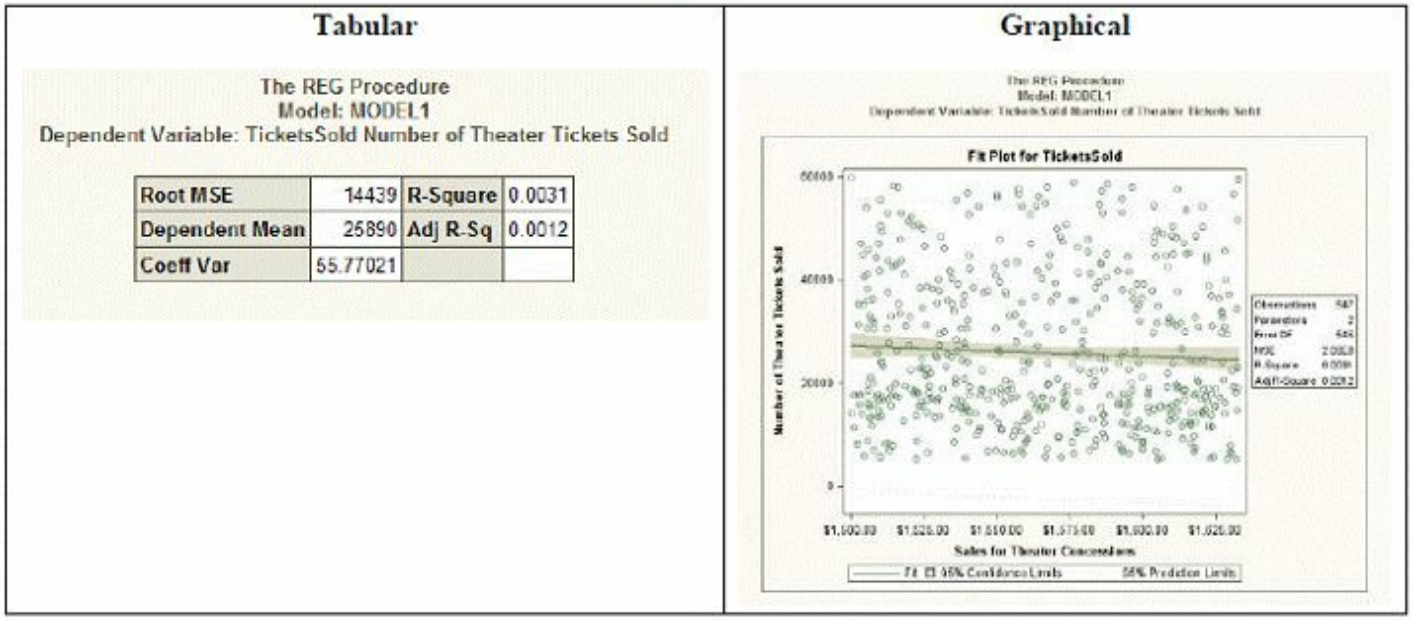


Figure 20.6



Tabular

The REG Procedure
Model: MODEL1
Dependent Variable: TicketsSold Number of Theater Tickets Sold

Root MSE	14439	R-Square	0.0031
Dependent Mean	25890	Adj R-Sq	0.0012
Coeff Var	55.77021		

Graphical

Figure 20.7

FitPlot.png	PNG Image
FitPlot1.png	PNG Image
PPPlot1.png	PNG Image
BoxPlot.png	PNG Image
BoxPlot1.png	PNG Image
PPPlot.png	PNG Image

Creating Graphics Output with the Statistical Graphics (SG) Procedures

Automatic graph creation is just one component of the ODS Statistical Graphics. You can also use the statistical graphics (SG) procedures beginning in SAS/GRAPH 9.2: SGPLOT, SGSCATTER and SGPANEL. There's enough material about the SG procedures to fill a book. The scope of this book limits us to only a few examples. This example illustrates some simple applications of SGPLOT and SGPANEL.

With the SGPLOT procedure, you can create one or more plots, or you can overlay plots on the same set of axes. The statistical graphs that PROC SGPLOT can create include box-and-whisker plots, histograms, regression plots, and line plots. The VBOX statement creates a vertical box-and-whisker plot in which the distribution of data, outliers, median, and mean are all displayed on one graph.

The SGPANEL procedure creates a panel of graph cells for the values of one or more classification variables. It supports the same VBOX statement as PROC SGPLOT to produce box-and-whisker plots. Specifying a classification variable with the PANELBY statement creates a cell in a panel of graphs for each value of the variable. In conjunction with the VBOX statement, the PANELBY statement can generate several box-and-whisker plots in a lattice format for easy comparison.

Continuing from the analysis of the data set MOVIEPLOTS in the previous example, imagine that your boss really liked the box-and-whisker plots from PROC ANOVA. He wants to see a similar type of plot for ticket sales by year and theme. We do not need to repeat the analysis of variance and can instead just use the VBOX statement in SGPLOT and SGPANEL. The PROC SGPLOT step that follows summarizes the sales for each year with one box-and-whisker plot. The PROC SGPANEL places the yearly ticket sales for each theme into a lattice format.

```
ODS HTML PATH='.' (URL=NONE)
        FILE='SalesPlots.html' STYLE=EGDEFAULT;

proc sgplot data=movieplots;
  vbox ticketssold / category=year;
  title 'PROC SGPLOT Output';
run;

proc sgpanel data=movieplots;
  panelby theme;
```

```
  vbox ticketssold / category=year;
  title 'PROC SGPANEL Output';
run;
ODS _ALL_ CLOSE;
```

Figure 20.8 displays the HTML output produced by the SGPLOT procedure. Figure 20.9 displays the results of the SGPANEL procedure.

The SGPANEL procedure enables you to place statistical graphs side by side for comparison purposes. This enables your graphical output to have the look of "small multiples," as described by graphical presentation experts such as Edward Tufte or Stephen Few.

Figure 20.8

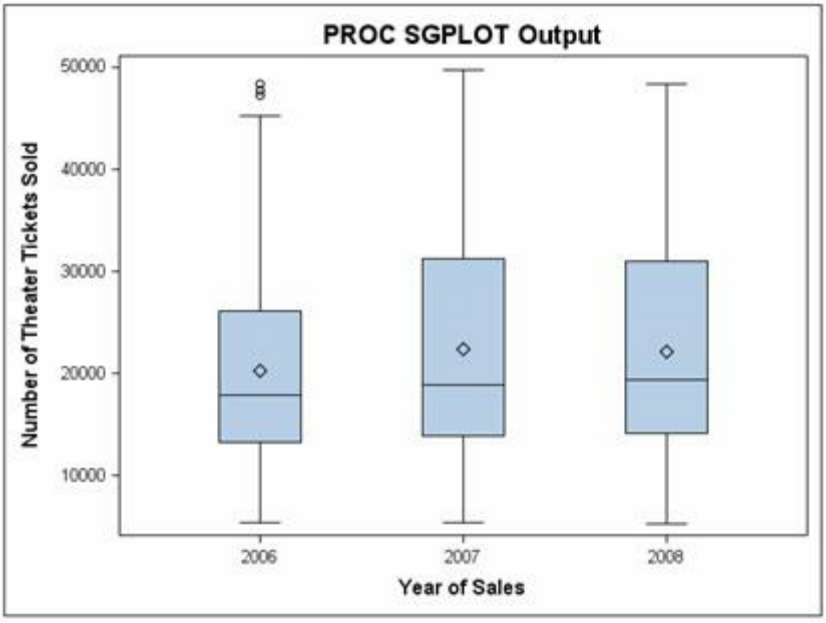
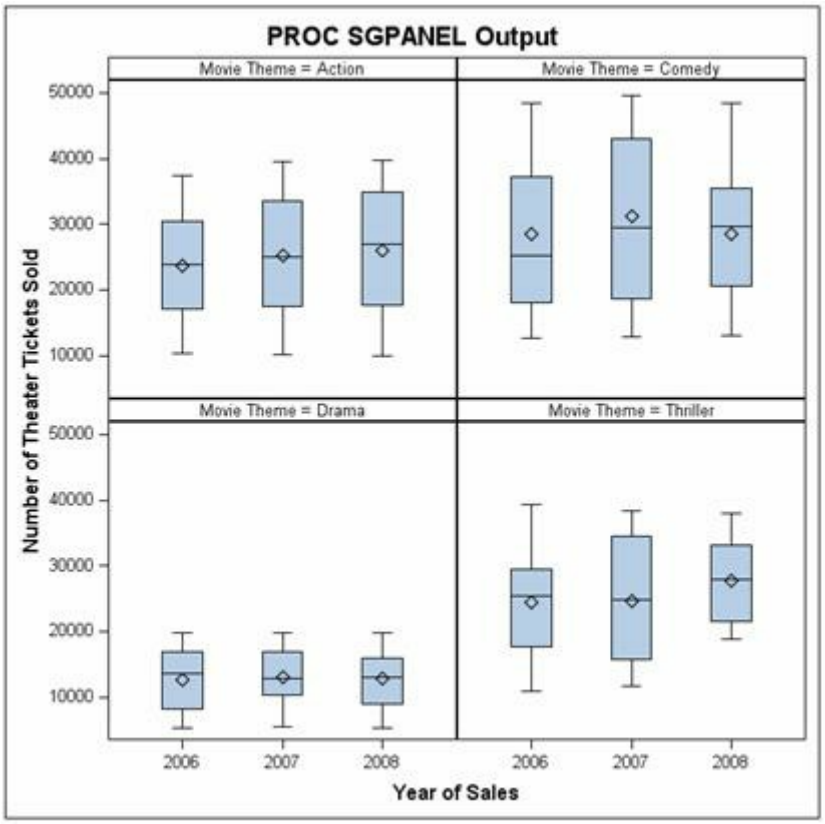


Figure 20.9


PROC SGPANEL Output
Movie Theme = Action
Movie Theme = Comedy
Movie Theme = Drama
Movie Theme = Thriller
Number of Theater Tickets Sold
50000
40000
30000
20000
10000
2006
2007
2008
Year of Sales

Creating Output Using Graph Templates and the SGRENDER Procedure

The last component of the ODS Statistical Graphics framework to cover is the Graph Template Language (GTL). GTL is used only with template-based graphics. If you want to alter the output created by PROC ANOVA, for example, you could use the ODS TRACE ON statement to determine the name of graph templates used in creating your graphics output, just as we used ODS TRACE to find out the name of table templates for tabular output. Then you could modify the GTL code to create a changed template for use. However, instead of modifying the graph from a procedure, the next example creates a graph template that PROC SGRENDER uses to construct our graphs.

Your boss decides that he likes the look of the side-by-side graphs from the previous example. Now he wants to see a bar chart of themes with a chart for sales and a chart for concessions side-by-side for each year, using the default style for SAS Enterprise Guide. The key to making this work is using the LAYOUT statement in GTL to define areas in a graph.

Think of the graph area as a box. There are many ways to subdivide the box. Our example starts with the DEFINE statement that creates a STATGRAPH template named COMPARE. The LAYOUT GRIDDED statement defines a gridded layout that divides the graph area into two columns. The first LAYOUT OVERLAY statement places the sales bar chart into one column. The next LAYOUT OVERLAY statement places the concessions bar chart in the second column. The BARCHART statement within each of the LAYOUT OVERLAY sections or blocks specifies the variables to place on the X-axis and Y-axis for each bar chart. The XAXISOPTS and YAXISOPTS options label the axes. After you have the basic layout the way you want, you can then enhance the graph, perhaps to add more padding around the two images or to add creation date information or other text.

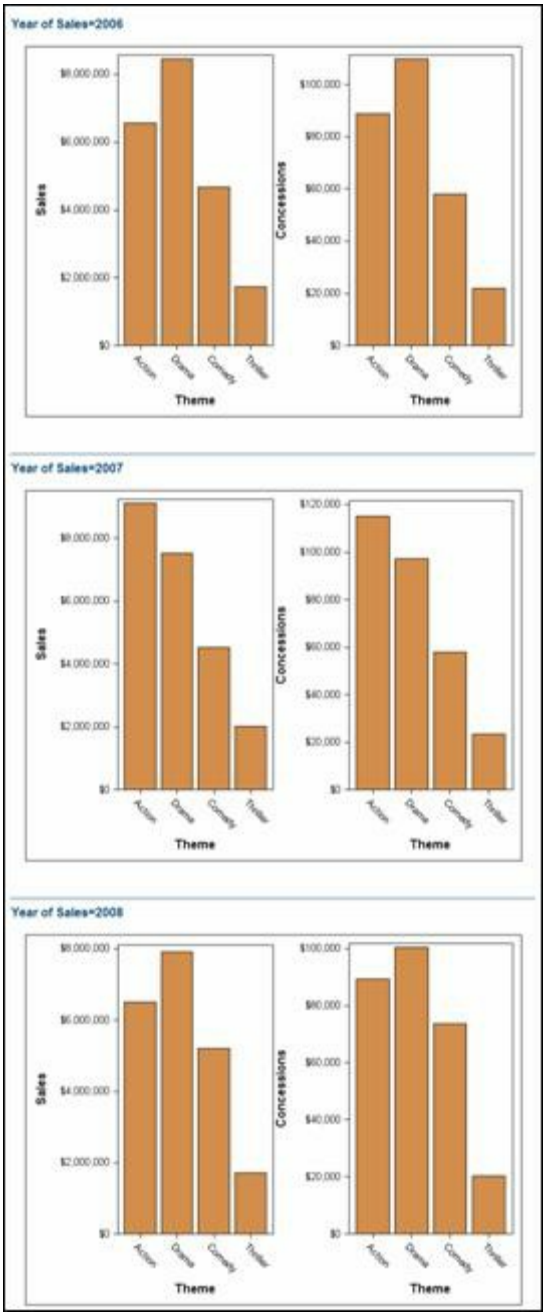
The SGRENDER procedure, with the BY statement, uses the new template COMPARE to render the side-by-side charts. If the BY statement were omitted, then only one pair of graphs would be created, which would represent the figures across all years.

```
proc template;
  define statgraph compare;
    begingraph;
      layout gridded / columns=2;
        layout overlay / xaxisopts=(label="Theme")
                         yaxisopts=(label="Sales")
                         cycleattrs=true;
          barchart x=theme y=Sales;
        endlayout;
```

```
        layout overlay / xaxisopts=(label="Theme")
                         yaxisopts=(label="Concessions")
                         cycleattrs=true;
          barchart x=theme y=Concessions;
        endlayout;
      endlayout;
    endgraph;
  end;
run;

ODS HTML PATH='.' (URL=NONE)
        FILE='ThemeStats.html' STYLE=styles.egdefault;
proc sgrender data=movieplots template=compare;
  by year;
run;
ODS HTML CLOSE;
```

Figure 20.10 shows the results of the SGRENDER procedure. Each row shows a pair of charts for one year.

Figure 20.10


Year of Sales=2006
Sales
Concessions
Theme
Action
Drama
Comedy
Thriller
Year of Sales=2007
Year of Sales=2008

Creating HTML Output with SAS/GRAPH

Creating HTML output from SAS/GRAPH is easy. All you have to do is write your usual SAS/GRAPH code and wrap ODS invocation statements around it. You might or might not need a GOPTIONS statement, because SAS 9.2 uses default devices for each destination.

You need an ODS HTML statement with at least a FILE= or BODY= option to name the file you are creating. You also need a PATH= option to point to the location where the body file and graphic images should be stored. When you use the PATH= option, the URL=NONE suboption prevents insertion of the physical storage path information in the <IMG> tag. The following example places the body file and the graphics file in the C:\temp directory.

Two items to keep in mind as you write your ODS graphics code are for selection of the device driver and selection of a SAS style template. The default device driver for ODS HTML in SAS 9.2 is the PNG driver. If you want to change from the PNG driver, explicitly specify the device driver with the GOPTIONS statement and the DEVICE= option, as shown in the following example. The driver you select must be compatible with ODS HTML. With GSTYLE in effect (the default), the SAS style template is used in graph creation. If you do not want a SAS style template used for your graphics output, set the NOGSTYLE option. Figures 20.11 and 20.12 show the differences between having GSTYLE or NOGSTYLE in effect.

```
ODS LISTING CLOSE;
ODS HTML PATH='c:\temp' (URL=NONE)
        BODY='Pies.html'
        STYLE=ANALYSIS;
GOPTIONS DEVICE=GIF BORDER;
title 'Pie Chart: Annual Sales by Flavor';
proc gchart data=PieStats;
  pie PieType / sumvar=Sales
            other=0
            midpoints='Blueberry' 'Apple' 'Cherry'
                  'Banana Creme' 'Pecan' 'Lemon Meringue'
            value=none
            percent=arrow
            slice=arrow
            noheading;
run; quit;
```

```
ODS HTML CLOSE;
```

One way to save time and system resources is to close the LISTING destination before creating your ODS graphical output. With the LISTING destination closed, your graph just goes directly to an external file. The output file has a file extension based on the DEVICE= option that you specified or the default device for a specific destination.

Figure 20.11 shows the HTML page produced by the preceding code. The <IMG> tag inside the HTML file points to the GIF file created by the GCHART step. Note that the Analysis style template determines the color of the pie slices. In addition, the SAS title is contained within the border of the graph area, which indicates that SAS/GRAPH controlled the title. If you reran the preceding job and the only modification you made was to replace GSTYLE with NOGSTYLE, SAS would produce your graph with the default SAS/GRAPH colors and fonts as shown in Figure 20.12.

We discuss device drivers and other destinations in an upcoming topic. In the next example, we discuss how other simple changes can have an impact on the graphic output that you create with ODS.

Figure 20.11

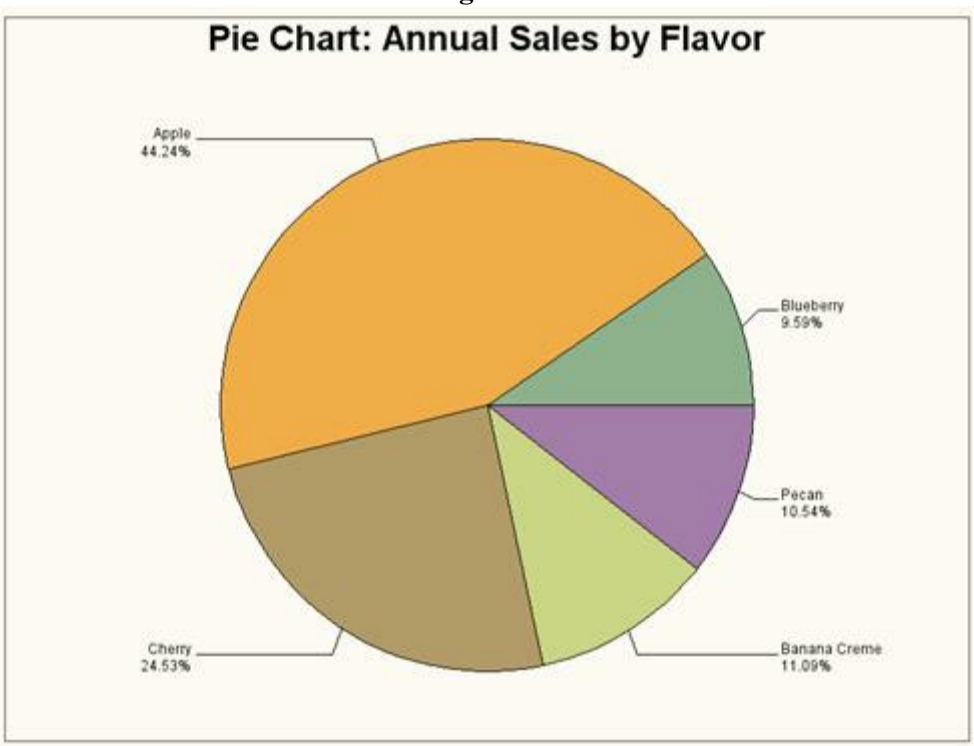
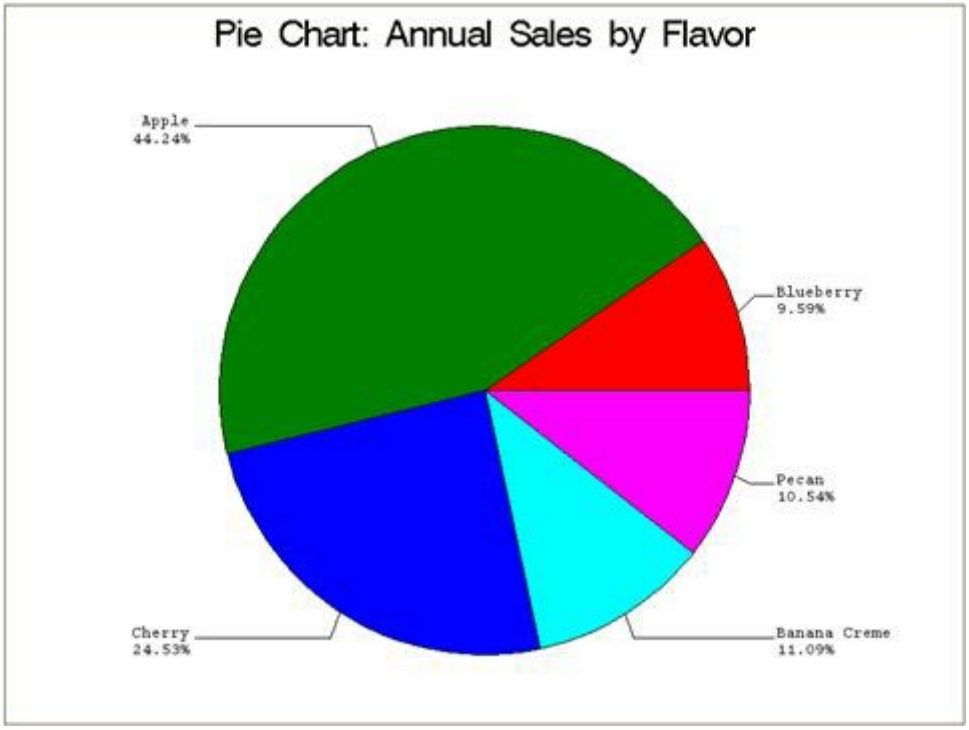


Figure 20.12


Pie Chart: Annual Sales by Flavor
Apple
44.24%
Blueberry
9.59%
Pecan
10.54%
Banana Creme
11.09%
Cherry
24.53%

Using ODS Options and SAS/GRAPH Options to Change Output

Many SAS/GRAPH GOPTIONS and procedure options can affect colors, line styles, fonts, and other style characteristics. When the SAS style template is being used, your SAS/GRAPH options and code can override what is specified in the style template. In addition, some ODS destinations have options that can affect the graphic output.

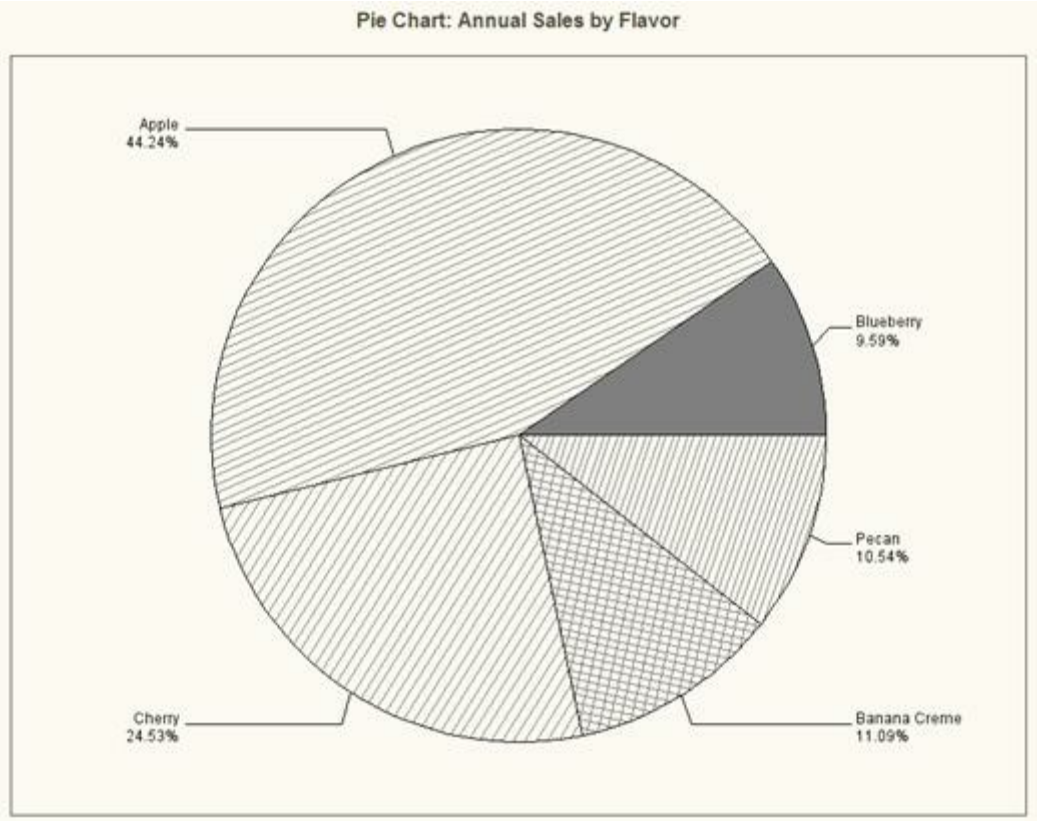
The following code illustrates both of these scenarios. The program is the same as that in the previous example, with two changes. First, the addition of the ODS NOGTITLE option on the ODS HTML statement causes SAS to place the title text outside the graph area boundaries. When NOGTITLE is in effect, the title attributes are controlled by the SystemTitle elements of the ODS style template being used. When GTITLE is in effect, the title attributes are governed by the GraphTitleText element. The second change in the code is the use of the CFILL= option on the PIE statement, which assigns gray as the fill color of the slices.

```
ODS HTML PATH='c:\temp' (URL=NONE)
                BODY='Pies.html' NOGTITLE
        STYLE=ANALYSIS;

GOPTIONS DEVICE=GIF BORDER;
title 'Pie Chart: Annual Sales by Flavor';
proc gchart data=PieStats;
  pie PieType / sumvar=Sales
            other=0
            midpoints='Blueberry' 'Apple' 'Cherry'
                  'Banana Creme' 'Pecan' 'Lemon Meringue'
            cfill=gray
            value=none
            percent=arrow
            slice=arrow
            noheading;
run;
quit;
ODS HTML CLOSE;
```

If you compare Figure 20.13 to the previous example's output in Figure 20.11, you can see that these changes have made a significant impact on the appearance of the graph output.

For more information about using SAS/GRAPH GOPTIONS, global statements, and procedure statements to override style template defaults, see the topic "Controlling the Appearance of Your Graphs" in the *SAS/GRAPH 9.2 Reference*. In particular, the subtopics entitled "Overriding Style Attributes with SAS/GRAPH Statement Options" and "Graphical Style Element Reference for Device-Based Graphics" will help you decide how and when to change a style template versus using SAS/GRAPH methods.

Figure 20.13



If you open your HTML page and the graph is not displayed, check the PATH= option. If you move the graph to a different location, the HTML page can no longer find its graphic. The best way to create graphics is to create a local directory structure that mimics the directory structure on your Web server. Then, if the file loads on your local machine, you know that it will also load on the Web server machine.

Understanding Devices and Destinations

As we've seen in other discussions of ODS destinations and style templates, each destination has a default style template that is used in the creation of output. So too does each destination have an appropriate or preferred image format and a default resolution or dots-per-inch (DPI) value that is used in the creation of ODS output files.

With the ODS HTML destination, external image files are created when you use SAS/GRAPH and SAS/STAT procedures that produce graphics. However, other destinations, such as RTF and PDF, do not use external tags or pointers to a graphic file. Instead, graphics images sent to these destinations are internally converted and embedded in the RTF or PDF file.

Table 20.2 lists default settings for graphics output for several ODS destinations.

Table 20.2: Default Settings for Graphics Output for ODS Destinations

Destination	Default Format	Default DPI	Default Style
LISTING	PNG	100	Listing
HTML	PNG	100	Default
RTF	Image is embedded in the file.	200	RTF
TAGSETS.RTF	Embedded image	200	RTF
Printer Family (PS,PDF)	Image is embedded in the file.	150	Printer
PCL	Image is embedded in the file.	150	monochromePrinter

In versions of SAS/GRAPH before SAS 9.2, only ActiveX and Java device drivers used ODS style templates. However, now that SAS 9.2 by default uses style templates for all graphical output, you might only need to specify explicit device options if you have a particular need for a device format.

The SAS Technical Support Web site has several documents in which they discuss the pros and cons of various device drivers in SAS 8 and SAS®9. Starting in SAS 9.2, the recommendation is that you use either the PNG driver or the SASEMF driver for RTF graphic output creation, and use SASPRTC, PNG, PDF, or PDFC drivers for PDF graphic output creation. Refer to Tech Support Usage Notes 35957, 32474, and 19331 for specific information about RTF and PDF device recommendations.

The SAS 9.2 default driver for the RTF destination is the SASEMF driver, for the PDF destination it is SASPRTC, and for the HTML destination it is PNG. Figure 20.14 shows how output differs when you send the same graph to the three destinations using the default drivers. The same GCHART program that was used to create Figure 20.1 was executed once for each of three destinations, so we are not repeating the code here. Figure 20.15 shows how the graphs look when you send output to the RTF and PDF destinations using the PNG driver. In

the next topic, we look at explicitly specifying other drivers for these destinations.

Here is the <IMG> tag created for the ODS HTML output in Figure 20.14:

```
<img src="def.png" style="border-width:0px; height: 600px; width: 800px;">
```

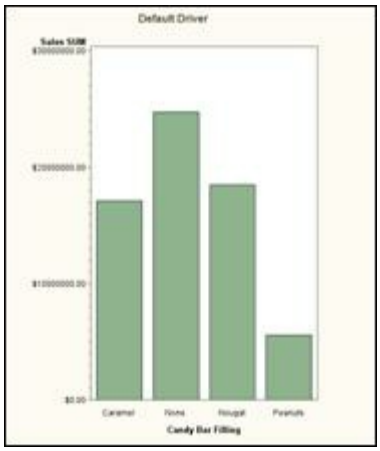
Although we can't look inside the PDF file, we can look inside the RTF file and search for the \PICT instruction: {\pict\emfblip\picwgoal8640\pichgoal5760 .

The specification of emfblip as the value for the PICTTYPE control instruction shows that the default EMF device was used to create this graphic image. On the other hand, if we look for the \PICT instruction in the file created by the PNG driver, we see this instruction: {\pict\pngblip\picwgoal11520\pichgoal8640 . This instruction specifies pngblip, which indicates that the embedded graphic image is in PNG format. The sizes specified in the PICT control word are the image size in twips (1/1440 of an inch).

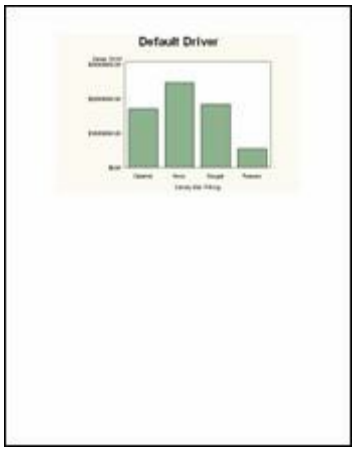
Notice the size differences of the bar charts in Figures 20.14 and 20.15. The outputs show that you cannot assume that the size of the graphic output will be the same in all destinations. You have to test your output with either the default device or a different device and set your SAS/GRAPH sizing options (HSIZE, VSIZE, XPIXELS, YPIXELS, XMAX, YMAX) accordingly. Consult the SAS/GRAPH documentation if you are not familiar with the use of these GOPTIONS.

Figure 20.14: Default Driver for All Destinations

HTML (Browser View)
PNG



RTF (Print Preview)
Embedded



PDF (Print Preview)
Embedded

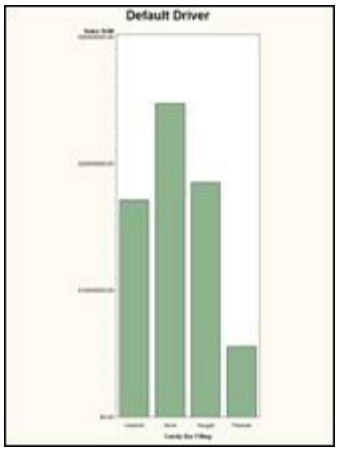
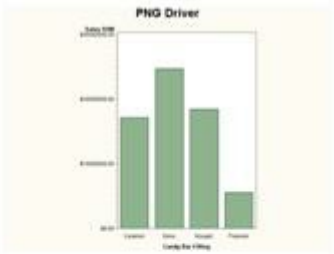
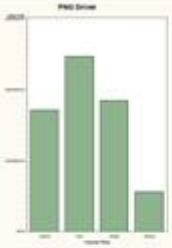


Figure 20.15: PNG Driver for PDF and RTF Destinations

RTF (Print Preview)
Embedded

PDF (Print Preview)
Embedded


PNG Driver



More about Devices and Destinations

In the previous topic, the code that was submitted used either the default device driver for the destination or it used the PNG driver. The DEVICE= graphics option is the way you specify which device driver to use, as shown in the following code. The DEVICE=JPEG setting applies to both the PDF and RTF files shown in Figure 20.16.

```
ODS PDF FILE='Output_20_16.pdf' STYLE=ANALYSIS;

ODS RTF FILE='Output_20_16.rtf' STYLE=ANALYSIS;

GOPTIONS DEVICE=JPEG;

proc gchart data=CandyBars;

  vbar Filling / sumvar=Sales;

run; quit;

ODS _ALL_ CLOSE;
```

This same PROC GCHART program created the charts in Figures 20.16 and 20.17. The ACTXIMG driver was used for Figure 20.17 for all three destinations. The only change to the code to produce Figure 20.17 is the following:

```
GOPTIONS DEVICE=ACTXIMG;
```

There is not much difference in the appearance of the outputs from these drivers in the ODS destinations that support style. In the end, if the results from the default device do not suit your needs, you have extensive choices for device types. Refer to the documentation topic "Summary of Default Destinations, Styles, and Devices," for a comprehensive list of which devices are recommended for each ODS destination.

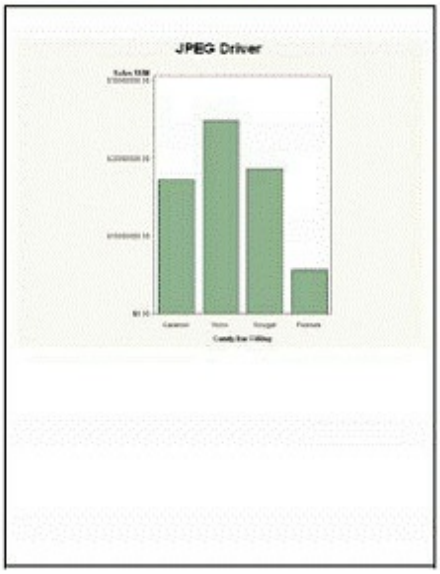
If you are running legacy SAS/GRAPH code, you will need to refer to the documentation topic "Changing the Appearance of Output to Match That of Earlier SAS Releases," which outlines the changes you will want to make to your code. First, you will want to specify the NOGSTYLE option, as previously demonstrated. Second, for complete compatibility with previous releases of SAS, you might want to specify the "Z" version of the driver, such as ZGIF, ZJPEG or ZPNG. The "Z" versions perform font rendering using the prior device methods. You might also need to specify HOST_PIXELS as the value of the FONTRENDERING= option to get the prior font-rendering behavior. This option specifies whether devices that are based on the SASGDGIF, SASGDTIF, and SASGDIMG modules render fonts by using the operating system or by using the FreeType engine.

Rather than continuing with a discussion of static device drivers with ODS destinations, our next topics focus on generating drill-down graphs using ODS HTML and interactive graphs using ActiveX and Java drivers.

Figure 20.16: Using JPEG Drivers

```
GOPTIONS DEVICE=JPEG;
```

RTF (Print Preview)
Embedded



PDF (Print Preview)
Embedded

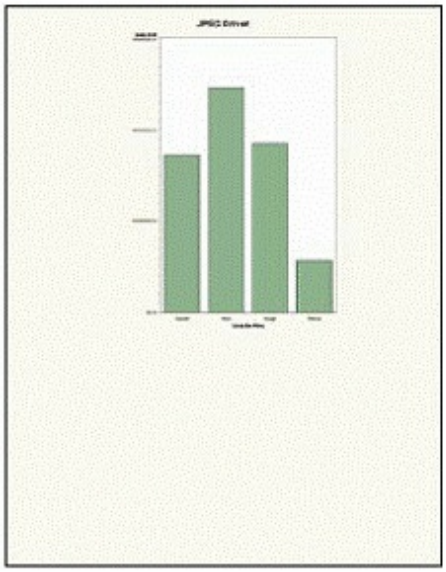
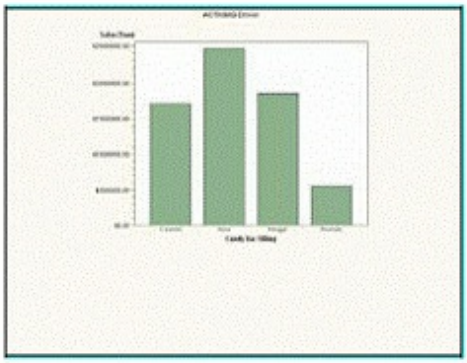


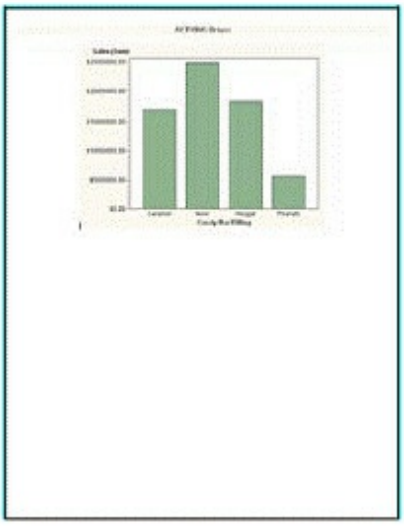
Figure 20.17: Using ACTXIMG Driver for All Destinations

```
GOPTIONS DEVICE=ACTXIMG;
```

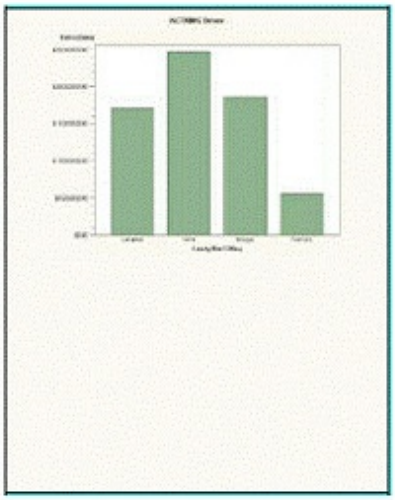
HTML (Browser View)
JPEG



RTF (Print Preview)
Embedded



PDF (Print Preview)
Embedded



Creating HTML Output: Graphs as GIFs or JPEGs

Drill-down graphs in HTML use a technique that enables you to present a summary graph and include the capability to present detailed data for interested viewers. The way it works is that you create a graph with HTML "hot spots." When you click on one of these "hot spots," your browser links to the detail data associated with that part of the graph. This example gives a basic introduction to creating drill-down graphs. More detailed information is available in the ODS online documentation.

The first step is to create a data set with a variable to hold the location of the HTML pages with the detail information. In this case, it is a single page with a separate named anchor (pie1, pie2, ...) for each detail table.

```
    Data PieStatsDrill;
  set PieStats;
  format rpt $40.;
  if PieType='Apple' then rpt='href="DrillDowns.htm#pie1"';
  else if PieType='Banana Creme' then rpt='href="DrillDowns.htm#pie2"';
  else if PieType='Blueberry' then rpt='href="DrillDowns.htm#pie3"';
  else if PieType='Cherry' then rpt='href="DrillDowns.htm#pie4"';
  else if PieType='Lemon Meringue' then rpt='href="DrillDowns.htm#pie5"';
  else if PieType='Pecan' then rpt='href="DrillDowns.htm#pie6"';
run;
```

The second step is to create the detail pages. Notice the ANCHOR= option that creates the anchors used in the preceding code.

```
    ODS HTML BODY='c:\temp\DrillDowns.htm' ANCHOR='pie1' STYLE=ANALYSIS;
proc means data=PieStatsDrill mean nonobs maxdec=0;
  by PieType;
  class year;
  var Sales;
run;
ODS HTML CLOSE;
```

The final step is to create the summary graph that is used for drilling down. The only change here is the addition of the HTML= option on the PIE statement. This points to the variable that holds the link location for each pie slice. The resulting graph is shown in Figure 20.18. When you click on the "Cherry" slice, the browser links you to the table shown in Figure 20.19.

```
    ODS HTML BODY='c:\temp\PieDrill.htm'
          STYLE=ANALYSIS;
title 'Pie Chart: Annual Sales by Flavor';
proc gchart data=PieStatsDrill;
  pie PieType / sumvar=sales
          percent=arrow slice=arrow value=none
          HTML=rpt;
run; quit;
ODS HTML CLOSE;
```

Figure 20.18

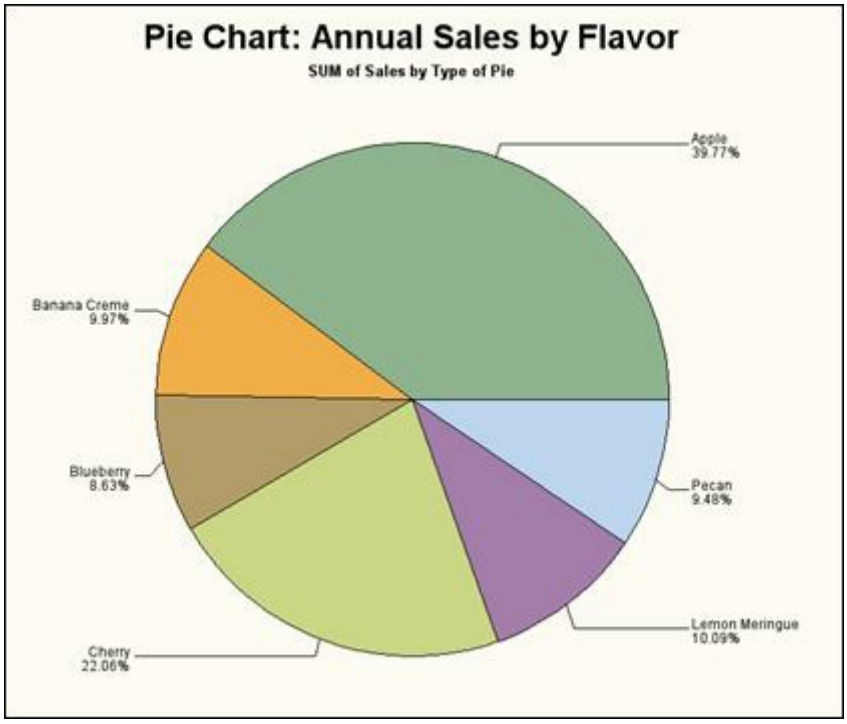


Figure 20.19

Type of Pie=Cherry

Analysis Variable : Sales	
Year of Sales	Mean
2006	118979
2007	118714
2008	114461

Although we have shown the drill-down capability with pie charts, this drill-down technique is supported in SAS/GRAPH for bar charts and plots as well. Consult the documentation topics "Enhancing Web Presentations" and "Adding Links with the HTML= and HTML_LEGEND= Options" for more examples of how to enhance your graphs with drill-down capabilities. The documentation for each SAS/GRAPH procedure indicates whether that procedure supports drill-down capability. For example, PROC GBARLINE supports drill-down functionality, but PROC GCONTOUR does not.

Creating HTML Output: Graphs as Java

If the viewers of your graphs want a high degree of interactivity, you might want to consider creating a Java graph instead of a static image such as that created with the GIF or PNG drivers. The JAVA device driver enables you to produce a graph that is actually a Java application. Not only can the graph be viewed, but graph attributes can be modified on the fly. Viewers can change the color or appearance, or they can drill down to more detailed information. In SAS 8.2, the Java driver created an <APPLET> tag; in SAS®9, the behavior of the Java driver changed and now an <OBJECT> tag is created.

To create a Java graph, submit the following code. You will be able to look at the graph, and anyone who has the right Java Runtime Environment and the right SAS/GRAPH Java Applets will be able to open the output HTML file and see the graph.

```
      ODS HTML PATH='c:\temp' (URL=NONE)
            BODY='JavaBar.htm' NOGTITLE STYLE=ANALYSIS;
GOPTIONS DEVICE=JAVA;
title 'Candy Bar Chart';
title2 'Annual Sales by Type of Chocolate and Filling';
proc gchart data=CandyBars;
  vbar3d Filling / sumvar=Sales
                 group=Chocolate;
run; quit;
ODS HTML CLOSE;
```

This code creates a single file, JavaBar.htm. Unlike GIF or PNG output, the graph is not in a separate file from the HTML page. Instead, the HTML contains code that is used by Java to generate the graph. The resulting graph is shown Figure 20.20. Notice the pop-up box that appears when the mouse hovers over a graph bar. This contains detailed information about the point being graphed.

The most powerful feature of Java graphs is what happens when you right-click anywhere on the graph. As shown in Figure 20.21, you get a pop-up menu that enables you to change the formatting of the graph. This example shows how you would change the type of chart. Other options enable you to change the colors or change the style of the bar. You can also click any of the bar groups to drill down to a graph of just that group. Creating Java graphs enables you to create a graph that can be customized as needed by your users. However, any changes you make interactively while running the applet are not retained in the HTML file.

Even though the Java graph information is all contained within the HTML file, some other components are needed to successfully deploy the Java graph on your Web server. The HTML file that you create contains the following HTML tags:

```
<PARAM NAME="JAVA_CODEBASE" VALUE="C:\Program Files\SAS\SASGraphJavaApplets\9.2">
<PARAM NAME="ARCHIVE" VALUE="graphapp.jar,xml1.jar,jgstex.jar">
```

These are the default values for SAS 9.2 Java graphics. In addition, a script is inserted in the HTML file that points to the download location for the JRE. When you are ready to deploy your graph, you need to know whether your consumers will access these files locally or whether they will be accessing the Java Applets and JAR files on the Web server.

If they will be accessing the files on the Web server, then you will need to use the CODEBASE= option to change the JAVA_CODEBASE parameter. This also means that you might need to enlist the help of your Web site administrator if the Java files will be stored on a local server, as described in the documentation topic "Specifying Parameters and Attributes for Java and ActiveX."

Figure 20.20

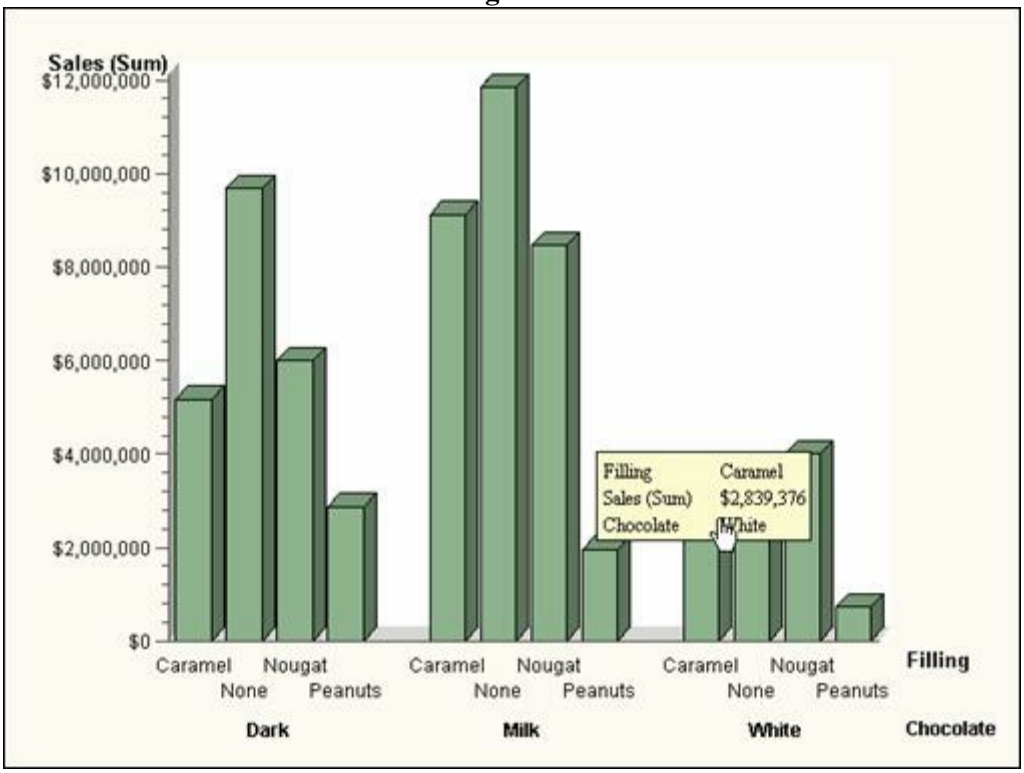
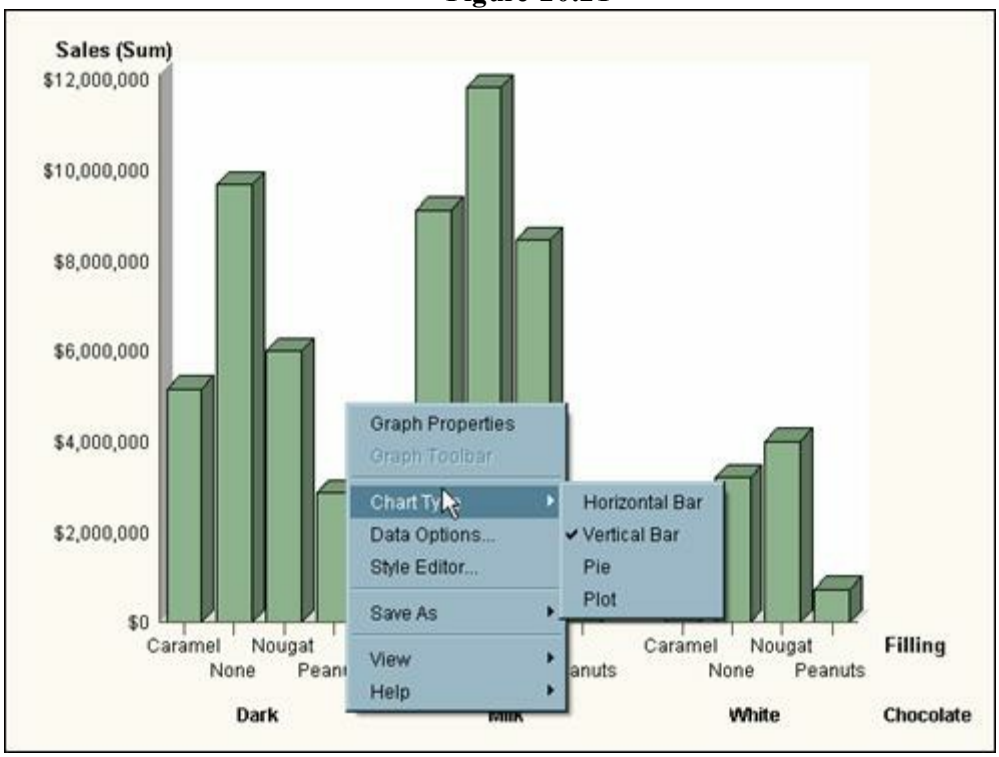


Figure 20.21



Creating HTML Output: Graphs as ActiveX

Another option for creating interactive graphs is to use the ACTIVEX device driver. This device driver uses Microsoft's ActiveX Control technology to create an interactive graph that can be viewed on a Web site or inserted into a Microsoft Office document. As with the previous example, if you have the correct ActiveX controls installed on your system, you will be able to view the graph created from this code:

```
      ODS HTML PATH='c:\temp' (URL=NONE)
            BODY='ActiveXBar.htm' NOGTITLE STYLE=ANALYSIS;
GOPTIONS DEVICE=ACTIVEX;
title 'Candy Bar Chart';
title2 'Annual Sales by Type of Chocolate and Filling';
proc gchart data=CandyBars;
  vbar3d Filling / sumvar=Sales
                 group=Chocolate;
run; quit;
ODS HTML CLOSE;
```

The resulting file is shown in Figure 20.22. Notice the pop-up box that appears when the mouse hovers over a graph bar. This contains detailed information about the point being graphed.

As with the Java graphs, the most powerful feature of ActiveX graphs is your ability to change the graph with a right-click anywhere on the graph. As shown in Figure 20.23, you get a pop-up menu that enables you to change the formatting of the graph. This example shows how you would change the type of chart. Other options enable you to change the colors or change the style of the bar. You can also click any of the bar groups to drill down to a graph of just that group. Creating ActiveX graphs enables you to create a graph that can be customized as needed by your users. Similar to the previous Java example, any change you make while running your graph interactively cannot be retained in the HTML file. However, if ActiveX is generated as RTF, then the changes you make in the graph can be saved in your Word document.

It is not required, but it is recommended that you remove any XPIXELS= or YPIXELS= options from your GOPTIONS statement. If you try to make an ActiveX graph smaller, you get a graph with a scroll bar that you have to use to see the entire graph.

When you create a Web presentation using the SAS/GRAPH ACTIVEX device, the resulting presentation is configured by default to prompt users to install the SAS/GRAPH ActiveX Control if it is not already installed. The installation program does not run if the control has already been installed. For more information about downloading and installing the ActiveX control, consult the documentation topic "Installing the SAS/GRAPH ActiveX Control" on the Technical Support Web site.

The same warning applies to both ActiveX and Java graphs. They both have some limitations to keep in mind. If pop-up windows, Java applets, or ActiveX controls are blocked at the browser, your users might not be able to view the graphs. Another limitation is that these drivers do not support all SAS/GRAPH procedures, and even for the procedures

they do support, they don't support all the options. Luckily, there is a documentation topic entitled "Troubleshooting Web Output" that outlines some common access problems and suggests solutions.

If you are only producing Web output, you might find the JAVA device easier to work with for you and your users. If you are creating a graph to import into a word processor, the ACTIVEX or the ACTXIMG driver might be a better choice.

Figure 20.22

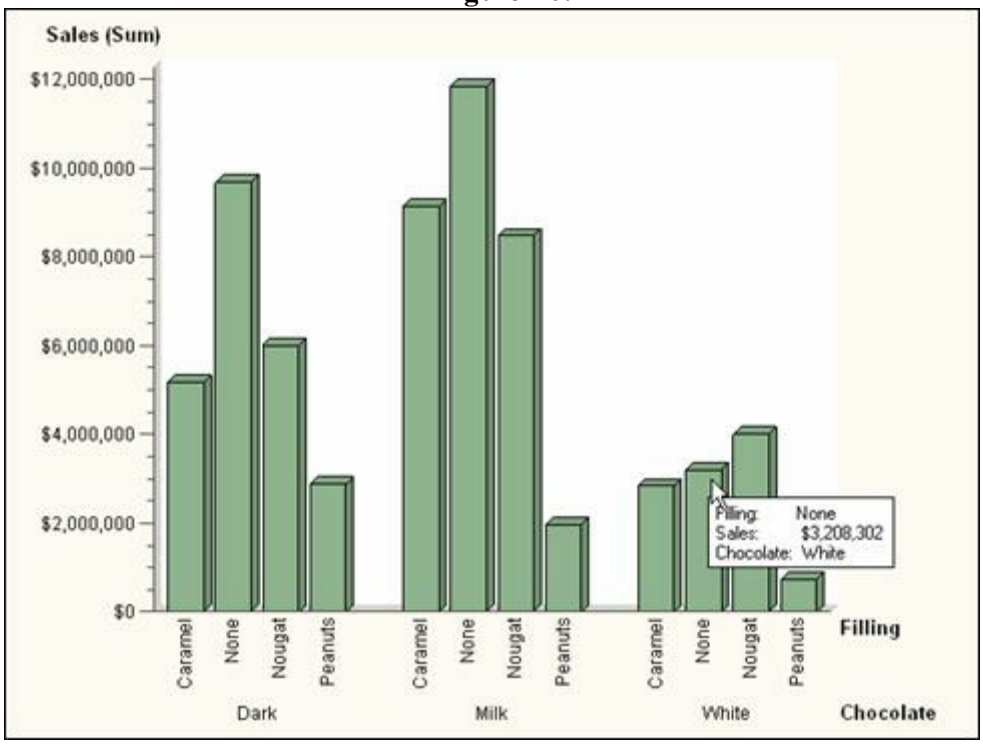
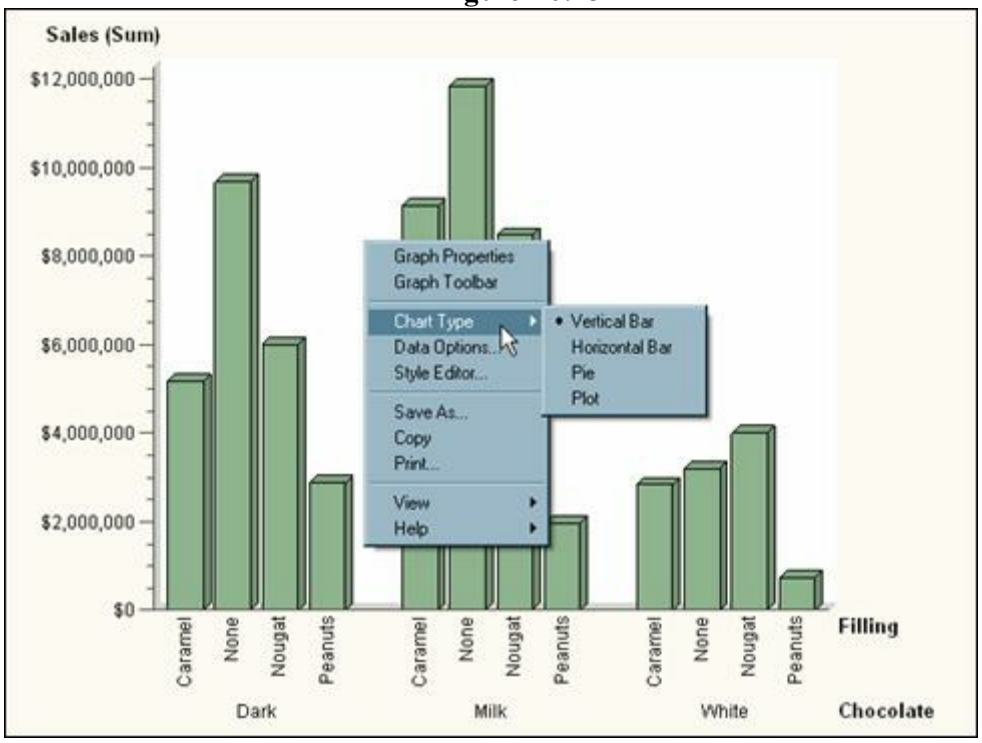


Figure 20.23



Creating RTF Output: Graphs as ActiveX

Adding a graph to your RTF output enables you to create impressive memos and reports. However, adding an interactive graph to a word processor document is even more impressive. This enables the readers of your memo or report to modify the graph to suit their needs.

The code for ActiveX output for the RTF destination is simple. You use the same ODS RTF statements as with any other procedure. Then you need to add the DEVICE=ACTIVEX option to your GOPTIONS statement.

```
      ODS RTF FILE='ActiveXPlot.rtf' STYLE=ANALYSIS;
GOPTIONS DEVICE=ACTIVEX;
title 'Movie Plots';
title2 'Ticket Sales by Theme';
proc gplot data=MoviePlots;
  plot Theaters*TicketsSold=Theme;
run; quit;
ODS RTF CLOSE;
```

When you open your RTF document, you will see the graph shown in Figure 20.24, assuming you have the ActiveX control installed locally.

This graph is fully interactive from within your word processor. When the mouse hovers over a point, you get the pop-up that describes the data point, as shown in Figure 20.24. If you right-click on the graph, you get a pop-up menu with options for modifying the graph, as shown in Figure 20.25.

ActiveX is an exciting new way to share SAS/GRAPH output, but for the ODS RTF destination, it comes with a number of critical limitations. First, it requires that you have the SAS/GRAPH ActiveX Control installed locally on your Windows computer. And anyone who opens your RTF file must also have the SAS/GRAPH ActiveX control installed locally. Unlike the HTML version of ActiveX output, there is no automated way to set up the driver in advance for the RTF user so it can be downloaded and installed as needed. That means all interactivity is lost if the ActiveX controls are not installed locally. For these reasons, we recommend that you consider using ActiveX with RTF only in situations where you are certain that the SAS/GRAPH ActiveX control is available. Otherwise, delivering content with ActiveX and ODS HTML might be a better choice.

Another limitation of using the ActiveX device with SAS/GRAPH is that it works with most SAS/GRAPH procedures, but not all. This limitation applies to both ODS HTML and ODS RTF. The procedures that it works with are G3D, GAREABAR, GBARLINE, GCHART, GCONTOUR, GMAP, GPLOT, GRADAR, GBARLINE, and GTILE. For a list of GOPTIONS that are supported, in addition to the specifics of procedure support, refer to the documentation topic “Summary of ActiveX and Java Support.”

The third limitation is that ActiveX is a Microsoft technology, and as such, it might only work correctly with Microsoft Office products. This means that you can open the file with Microsoft Word, and you can copy and paste the graph into Microsoft Excel and Microsoft

PowerPoint, but the graph might not work with other products or on any operating system other than a system based on Windows.

The fourth limitation is that many users might have ActiveX functionality turned off because it has proven to be a security risk. The technology can be used by hackers to insert a virus in your system. If you keep up with the latest security patches and anti-virus software, this should not be a problem. Nevertheless, many sites choose to disable ActiveX as a safety measure.

Figure 20.24

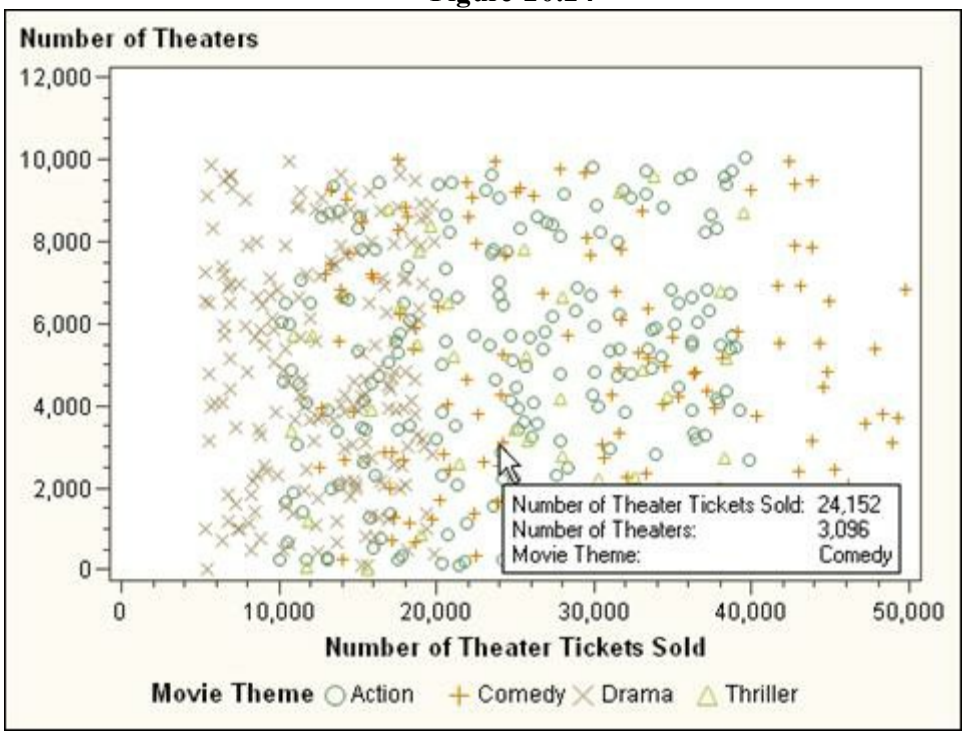
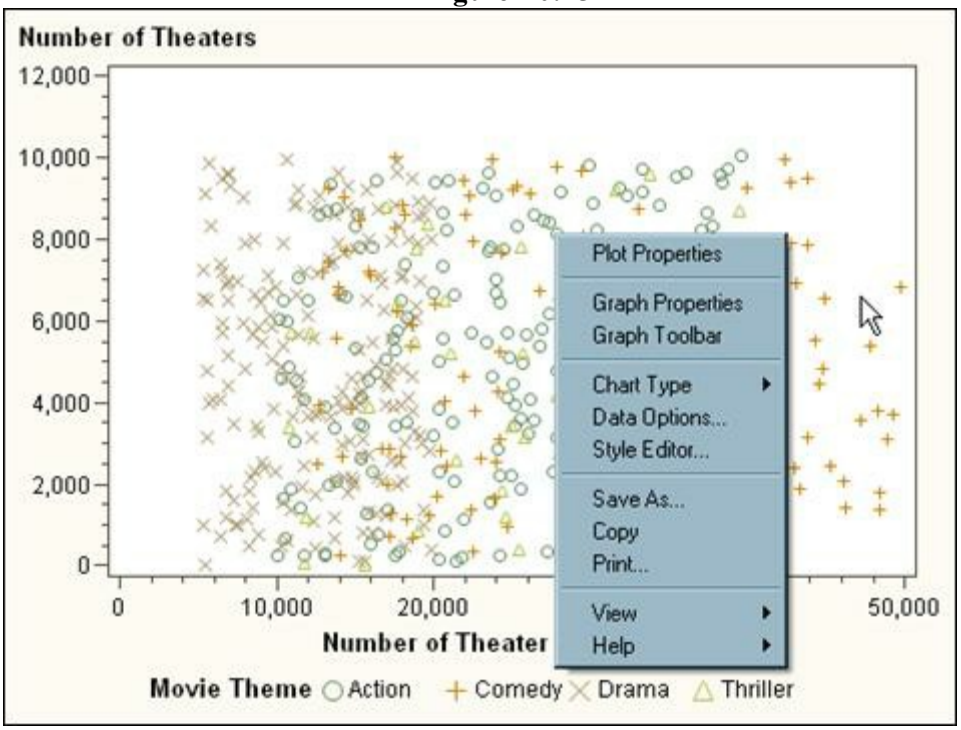


Figure 20.25



Combining Graphs with Text Output

With ODS, it is possible to combine the output from a number of procedures into a single output file. This means that you can create an HTML page with a table from PROC FREQ and a bar chart from PROC GCHART. In HTML, because there are no page breaks, the output appears to be on the same page. However, there are still large gaps between the procedures.

If you'd like to integrate your graphs and tabular output, there is a trick you can use to embed a graph in a row of a PROC REPORT table. The technique is simple. First, you need to create your graph and save it in a SAS catalog. The following code creates a bar chart called SalesBar and saves it in the WORK library under the catalog name CandyBarChart.

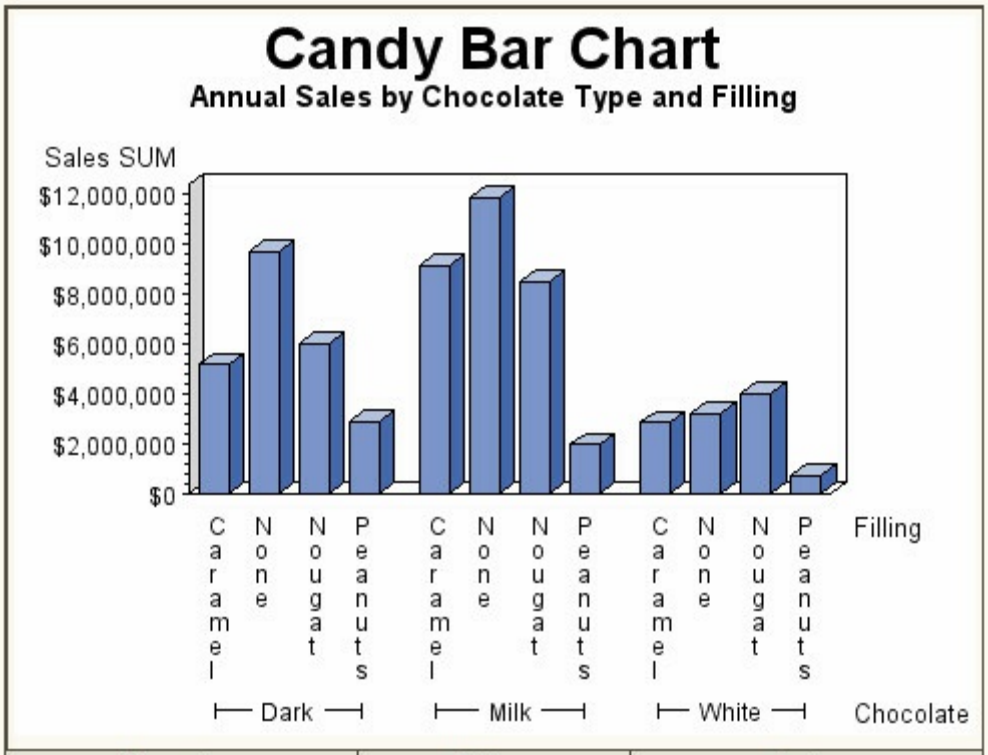
```
      GOPTIONS DEVICE=PNG XPIXELS=480 YPIXELS=360 GSFMODE=REPLACE;
title 'Candy Bar Chart';
title2 'Annual Sales by Chocolate Type and Filling';
proc gchart data=CandyBars GOUT=work.CandyBarChart;
  vbar3d Filling / name='SalesBar'
                 sumvar=Sales
                 group=Chocolate
                       raxis=0 to 12000000 by 2000000;
      run; quit;
```

The next step is to create a PROC REPORT table. After you have the basic table set up, you add a COMPUTE BEFORE _PAGE_ statement and a CALL DEFINE statement to insert a row at the top of the report and insert the graph you just created.

```
      ODS HTML BODY='CandyBarReport.html'
       PATH='c:\temp\' (URL=NONE)
            STYLE=ANALYSIS;
proc report data=CandyBars nowd;
  columns Chocolate Filling Sales;
  define Chocolate / group;
  define Filling / Group;
  define Sales / analysis sum format=dollar10.;
  compute before _page_ / center;
    call define( _ROW_, "GRSEG", "work.CandyBarChart.SalesBar" );
  endcomp;
run; quit;
ODS HTML CLOSE;
```

The resulting table is shown in Figure 20.26. The graph is now part of the tabular report, so both appear together on the Web page. This output could have been created by enclosing both the GCHART and REPORT procedures in ODS HTML statements and removing the GOUT= option and the COMPUTE block, but then there would be a gap between the chart and the table.

Figure 20.26



Chocolate	Filling	Sales
Dark	Caramel	$5,168,947
	None	$9,683,077
	Nougat	$6,004,396
	Peanuts	$2,876,484
Milk	Caramel	$9,131,443
	None	$11839215
	Nougat	$8,472,498
	Peanuts	$1,963,991
White	Caramel	$2,839,376
	None	$3,208,302
	Nougat	$4,008,680
	Peanuts	$729,681

Additional SAS/GRAPH and ODS Resources

The topic of SAS/GRAPH and ODS needs a book (or two) to really delve into all the possibilities for graph creation. You might have noticed that we don't have many hard-coded URL links in the book, because when Web sites are redesigned and updated, hardcoded links get broken. So, when we've referred to documentation topics, we intend for you to search support.sas.com or the internal SAS Help facility to search for those documentation topics, as shown in the screen shots on the next page.

Other useful resources are SAS Global Forum papers or user group papers on SAS/GRAPH topics. To get you started, we're providing this list of papers:

Table 20.3: Selected Papers on SAS/GRAPH and ODS Graphics Topics

Paper Title	Author and Paper Reference
Getting Started with ODS Statistical Graphics in SAS 9.2–Revised 2009	Robert N. Rodriguez (Paper 305-2008) SAS Global Forum 2008. The revised paper can be found on the SAS Web site: http://support.sas.com/rnd/app/papers/papers_da.html.
Modifying ODS Statistical Graphics Templates in SAS 9.2	Warren Kuhfeld (Paper 323-2009) SAS Global Forum 2009
Model Fitting and Data Analyses in SAS/ETS Software Using ODS Statistical Graphics	Jan Chvosta and Mark Little (Paper 243-2009) SAS Global Forum 2009
Secrets of the SG Procedures	Dan Heath (Paper 324-2009) SAS Global Forum 2009
GTL (Graphics Template Language) in SAS 9.2	Philip R Holland (Paper 218-2009) SAS Global Forum 2009
ODS Graphics Designer: An Interactive Tool for Creating Batchable Graphs	Sanjay Matange (Paper 331-2009) SAS Global Forum 2009
Clinical Trial Reporting Using SAS/GRAPH SG Procedures	Susan Schwartz (Paper 174-2009) SAS Global Forum 2009
Using PROC SGPLOT for Quick High-Quality Graphs	Lora D. Delwiche and Susan J. Slaughter (Paper 158-2009) SAS Global Forum 2009
Effective Graphics Made Simple Using SAS/GRAPH SG Procedures	Dan Heath (Paper 255-2008) SAS Global Forum 2008
ODS for the Professional Statistician	Steven M. LaLonde (Paper 377-2008) SAS Global Forum 2008
Gaining Power from GTL and ODS Style to Control Graphical Output	Lea Liu, Nan Guo and William F. McCarthy (Paper 092-2007) SAS Global Forum 2007
Using ODS Styles with SAS/GRAPH	Jeff Cartier and Dan Heath (Paper 088-2007) SAS Global Forum 2007
Creating Statistical Graphics in SAS 9.2: What Every Statistical User Should Know	Robert N. Rodriguez and Tonya E. Balan (Paper 192–31) SAS User Group International 31
A Programmer's Introduction to the Graphics Template Language	Jeff Cartier (Paper 262-31) SAS User Group International 31

Figure 20.27: Searching http://support.sas.com

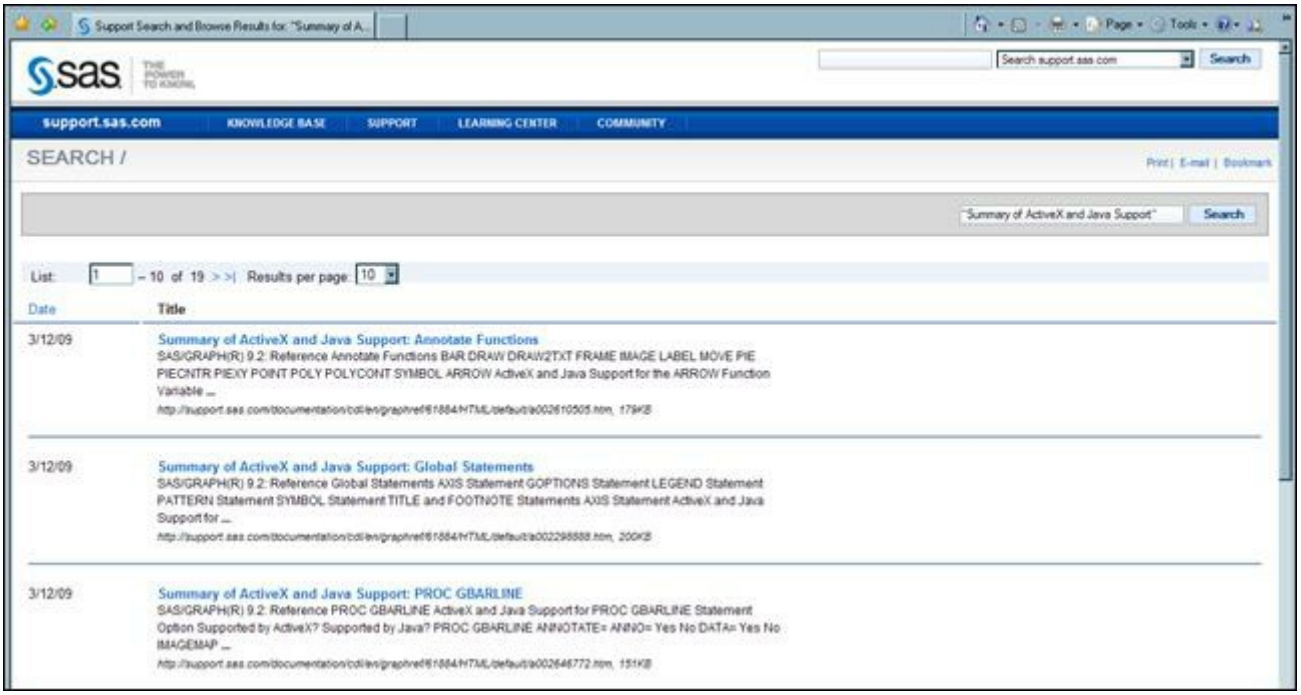
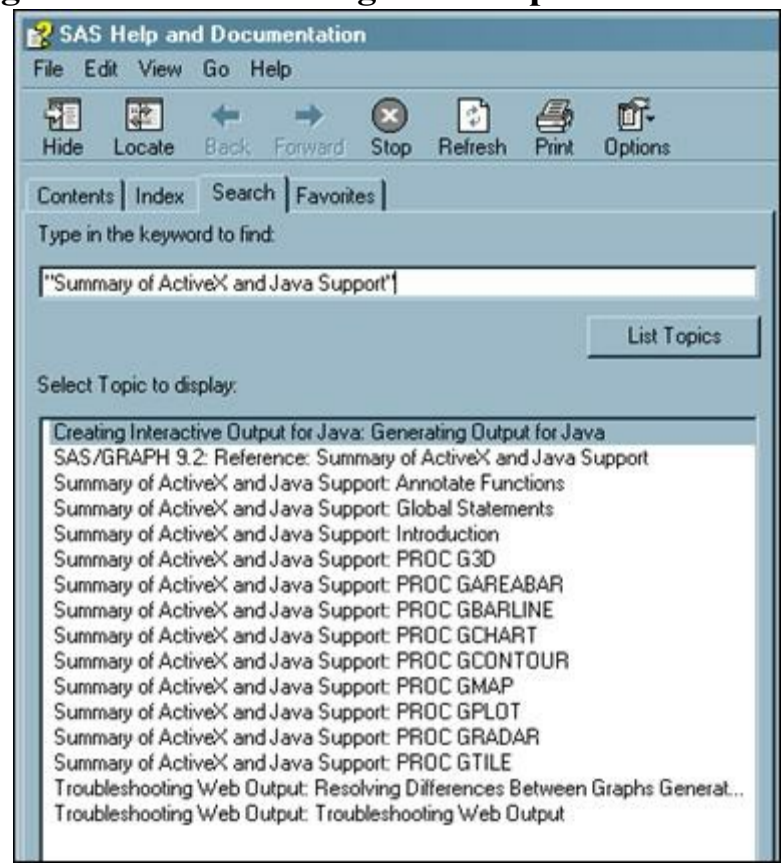


Figure 20.28: Searching SAS Help and Documentation



Appendix
Operating System Differences

If you don't use the Windows operating system, don't worry. Most of the code presented in previous chapters works just fine on other systems. ODS is fairly platform independent.

In fact, if you are using the ODS OUTPUT statement, you do not have to change a thing in order to run on any other platform. Just point your LIBNAME statement to the data location on your platform and you're ready to go.

Where the differences come in is with the filenames you specify for your output files in the HTML and RTF destinations, the identification of the PRINTER destination, and a few options needed for transferring files between systems.

This appendix starts with a basic example set up for the Windows operating system. The example is then repeated for other major operating systems, showing the modifications that are needed.

This chapter uses the data set CLINICAL to illustrate the concepts. This data set is available on the author pages for this book at http://support.sas.com/authors.

Setting Up the Example for Windows

There's nothing new in the code presented in this example. All of these features have been shown in previous chapters.

The example is a UNIVARIATE procedure that produces some statistics and a graph. The results are sent to HTML, RTF, and a printer. The code for this example in Windows is as follows:

```
ODS HTML FRAME="OSframe.html"
  BODY="OSbody.html"
  CONTENTS="OScontents.html
  PATH="c:\temp\" (URL=NONE);

ODS RTF FILE="c:\temp\OSfile.rtf";

ODS PDF FILE="c:\temp\OSfile.pdf";
proc univariate data=clinical;
   var age;
   symbol value=star;
   probplot age / normal(mu=est sigma=est) pctlminor;
run;

ODS _ALL_ CLOSE;
```

The ODS HTML statement requests that an HTML page be created with a table of contents. It specifies a file location on the local computer using PATH=, and adds the URL=NONE option to keep the path from being embedded in the links between the table of contents and the body page.

The ODS RTF statement specifies a file location on the local computer. This file can be opened directly from that location using a word processor.

The ODS PDF statement specifies a file location on the local computer. This file can be opened directly from that location using Adobe software.

Revising the Example for UNIX (HPUX, Sun, Solaris, Linux, etc.)

To run the code on the UNIX operating system, you need to make two changes. First, you need to change the pathnames to fit UNIX naming conventions. Second, you need to configure a UNIX printer.

```
ODS HTML FRAME="OSframe.html"
  BODY="OSbody.html"
  CONTENTS="OScontents.html"
  PATH="/tmp" (URL=NONE);

ODS RTF FILE="/tmp/OSfile.rtf";


ODS PDF FILE="/tmp/OSfile.pdf";

proc univariate data=clinical;
   var age;
   symbol value=star;
   probplot age /normal(mu=est sigma=est) pctlminor;
run;

ODS _ALL_ CLOSE;
```

The path naming changes are self-explanatory.

After you have made these changes, the HTML, RTF, and PDF files can be generated on your UNIX server.

Starting with SAS 9.2, you can view your HTML, PDF, and RTF output using the SAS Remote Browser. The SAS Remote Browser enables you to view some output in the default Web browser on your local computer without having to use FTP, e-mail, or any other transfer software. You might need to install the browser. SAS will prompt you with installation questions if it’s necessary.

Revising the Example for z/OS

To run the code on the z/OS host, you just need three changes. First, you need to specify a FILENAME statement for the output path. Second, the file naming conventions are slightly different and need to be modified. Third, you might need to add one option to the ODS HTML and ODS RTF statements.

You will have to decide where you want to view the output. Starting with SAS 9.2, you can view your HTML, PDF, and RTF output using the SAS Remote Browser. The SAS Remote Browser enables you to view some output in the default Web browser on your local computer without having to use FTP, e-mail, or any other transfer software. You might need to install the browser. SAS will prompt you with installation questions if it's necessary.

The first part of this example uses code to create a PDSE. The second part uses UFS (formerly HFS) to create the output files.

The first example creates EBCDIC HTML and RTF output that can be viewed locally with either the SAS Remote Browser or a z/OS Web server. PDF output is always created in ASCII.

```
      filename odsout '.temp.html'
              dsntype=library dsorg=po
              disp=(new, catlg, delete);
ODS HTML FRAME="OSframe"
       BODY="OSbody" (URL="OSbody.html")
       CONTENTS="OSconten" (URL="OSconten.html")
       PATH=ODSOUT (URL=NONE)
       RECORD_SEPARATOR=NONE;

      ODS RTF FILE=".temp.html(RTFfile)";

      ODS PDF FILE=".temp.html(PDFfile)";

      proc univariate data=clinical;
   var age;
   symbol value=star;
   probplot age /normal(mu=est sigma=est) pctlminor;
run;

ODS _ALL_ CLOSE;
```

The FILENAME statement sets up a PDSE for all of the output files. The ODSOUT fileref is then specified as the path. The names assigned to the individual output files have been changed to fit z/OS naming conventions. Two URL= options have been added to the BODY= and CONTENTS= options. These ensure that the links between the frame file, contents file, and the body file work correctly. The RECORD_SEPARATOR=NONE option specifies that ODS write one record at a time to the HTML file. This action prevents undesired wrapping of HTML tags so that the output can be viewed correctly with the SAS Remote Browser or the z/OS Web server.

If you intend to move your files to an ASCII-based operating system, you need to add the TRANTAB=ASCII option to your ODS HTML and ODS RTF statements as shown below and remove the RECORD_SEPARATOR= option from the ODS HTML statement. All non-PDF files created with ODS are generated as EBCDIC by default on the mainframe. The TRANTAB=ASCII option added to the ODS HTML and ODS RTF statements allows your HTML and RTF files to be copied over to an ASCII-based Web server.

```
filename ODSOUT ".temp.html"
 dsntype=library dsorg=po
 disp=(new, catlg, delete);
ODS HTML PATH=ODSOUT (URL=NONE)
 FRAME="OSframe"
 BODY="OSbody" (URL="OSbody.html")
 CONTENTS="OSconten" (URL="OSconten.html")
 TRANTAB=ASCII;

ODS RTF FILE=".temp.html(RTFfile)" TRANTAB=ASCII;

ODS PDF FILE=".temp.html(PDFfile)";
```

If you are running a Windows or UNIX Web server, use this code. When the files are transferred, they must retain their member names and add the .HTML extension in order for these URLs to be correct. Also, be sure to use a binary transfer to move the files to the Web server.

To use the RTF results, you need to copy the file to a system that has a word processor that can read RTF files. Be sure to use a binary copy and rename the file with an .RTF extension.

If you want to create files with UFS that can be viewed either with the SAS Remote Browser or the z/OS Web server, modify the sample code as follows.

```
options filesystem='hfs';

filename odsout '/u/userid/temp_html';

ODS HTML PATH=ODSOUT (URL=NONE)
      FRAME="OSframe.HTML"
 BODY="OSbody.HTML"
 CONTENTS="OSconten.HTML";
ODS RTF BODY="/u/userid/temp_html/RTFfile.RTF";

ODS PDF FILE="/u/userid/temp_html/PDFfile.PDF";
```

If you intend to view your output created with UFS on an ASCII Web server, you need to add the TRANTAB=ASCII option to your ODS HTML and ODS RTF statements.

```
ODS HTML PATH=ODSOUT (URL=NONE)
      FRAME="OSframe.HTML"
 BODY="OSbody.HTML"
 CONTENTS="OSconten.HTML"
      TRANTAB=ASCII;
```

```
ODS RTF BODY="/u/userid/temp_html/RTFfile.RTF" TRANTAB=ASCII;
```

Revising the Example for OpenVMS

To run the code on the OpenVMS operating system, you just need two changes. First, you need to set up a fileref for the path. Second, the file naming conventions are slightly different and need to be modified.

```
      filename odsout "device:[temp]";
ODS HTML FRAME="OSframe.html"
       BODY="OSbody.html"
       CONTENTS="OScontents.html"
       PATH=ODSOUT (URL=NONE);

ODS RTF FILE="device:[temp]OSfile.rtf";

      ODS PDF FILE="device:[temp]OSfile.pdf";

proc univariate data=clinical;
   var age;
   symbol value=star;
   probplot age /normal(mu=est sigma=est) pctlminor;
run;

ODS _ALL_ CLOSE;
```

The FILENAME statement sets up an appropriate location for all of the output files. This fileref is then used as the path. To use this example, replace “device” with the name of the device where you want your files to be stored. With this path in place, the names assigned to the HTML file are the same as for Windows. The names assigned to the RTF and PDF output follow OpenVMS naming conventions and include the device name.

Starting with SAS 9.2, you can view your HTML, PDF, and RTF output using the SAS Remote Browser. The SAS Remote Browser enables you to view some output in the default Web browser on your local computer without having to use FTP, e-mail, or any other transfer software. You might need to install the browser. SAS will prompt you with installation questions if it’s necessary.

Sources of Additional Information

This Appendix has provided a few simple examples of ODS on other operating systems. For more information, you can read the host-specific information in the SAS Companion for your operating system in the SAS OnlineDoc. Another source of assistance is the SAS customer support Web site, http://support.sas.com, which has links to many ODS examples and papers.

If you cannot find an answer to your question through these sources, try SAS Technical Support. First go to the Technical Support area on the SAS Web site and use the search option to see whether your answer is there. If not, e-mail or call Technical Support for further assistance.

Index

A

<A> tag
HREF= attribute 34–35, 52
table of contents and 52
TARGET= attribute 34–35, 52
abstract style elements 370–371
accessibility enhancements 452–453
Acrobat Distiller 192–193
Acrobat PDFWriter 192–193
ActiveX framework
creating HTML output 566–567
creating RTF output 568–569
dynamic navigation and 32
generating internal graphs 303
graphics output and 28, 544, 558–561
ACTXIMG device driver 303, 560
Adobe Acrobat software 192–193
Adobe Reader software 152, 154
AFTER DELETE option, OBPAGE statement (DOCUMENT) 310
ALERT function 374
aliases
defined 322
for color values 364
tagsets and 322
alignment
See justification style attribute
ALL option
ODS SELECT statement 242
TABLE statement, TABULATE procedure 522
ALT+F9 keystroke combination 102
ampersand (&) 333
ANALYSIS style template 554
Analysis style template 348
ANCHOR= option
FRAME= option, ODS HTML statement 72
ODS HTML statement 562
ODS PDF statement 166
anchor tags
drill-downs in output tables 66–67
linking results to Web sites 54–55
naming convention 54
overview 52–53
angle brackets (<>) 280, 321–323
annotations in PDF output 158–161
ANOVA procedure
creating graphic images 548
graphics output example 546–547
output objects and 200–203
APPEND option, ODS PATH statement 487
<APPLET> tag 564
ARCDATA macro variable 332

ATTACH= option, EMAIL engine 76
ATTRIB statement 264
AUTHOR= option, ODS RTF statement 120, 124
AUTOEXEC.SAS file 486
automatic macro variables 287

B

background color
modifying 442–443, 446–447, 519
style definitions 14, 16, 364–365
SYTLE_POPUP tagset 358–359
background images 458–459
BACKGROUND= style attribute
modifying with TABULATE procedure 520
traffic lighting technique 282, 502, 524
BACKGROUNDCOLOR= style attribute 358, 380, 386
BACKGROUNDIMAGE= style attribute 388, 458
backslash (\) 126, 134, 484
BARCHART statement 552
BASE table template 273
batch mode coding tips 298
BATCHFIXEDFONT= attribute 401, 404–405, 416
BEFORE_PAGE argument, COMPUTE statement (REPORT) 570
Beige style template 346
blank lines 510–511
BODY= option, ODS HTML statement 27, 30, 32
Body style element
adding background images 458
adding images to output 460–463
adding logo to output 456
defined 394
modifying attributes 378–379
running JavaScript programs 374
simulating page breaks 38
<BODY> tag 26–27
BODYTITLE_AUX option, ODS RTF statement 106
BODYTITLE option, ODS RTF statement
adding images to output 108
adding page numbering 100
headers/footers in MS Word 86
placing titles/footnotes 106
BOLD option
FOOTNOTE statement 56
TITLE statement 56
BOOKMARKGEN= option, ODS PDF statement 156
BOOKMARKLIST= option, ODS PDF statement 156
bookmarks
creating 154–155, 166–167
labels displaying in 154
modifying display of 156–157
BORDER= attribute 456
BORDERBOTTOMSTYLE= style attribute 384
BORDERCOLLAPSE= attribute 420
BORDERCOLOR= style attribute 380
BORDERLEFTSTYLE= style attribute 384
BORDERRIGHTSTYLE= style attribute 384
BORDERSPACING= style attribute 380–382, 394
BORDERSTYLE= style attribute 384

BORDERTOPSTYLE= style attribute 384
BORDERWIDTH= style attribute 380, 394, 420–423
BOTTOMMARGIN= system option 96, 186, 432
BOX option, PROC REPORT statement 512
BOX= option, TABLE statement (TABULATE) 516, 526
box plots 550
BREAK statement, REPORT procedure
SKIP option 510–511
STYLE= option 500
STYLESHEET= option 474
TITLE= suboption 27, 46–47, 80
URL= option 80
URL=NONE option 28, 74
Brown style template 346
browsers
See Web browsers
BULLET= attribute 428
BY-group processing
applying inline formatting 268
controlling output layout 262
listing BY groups 298–299
modifying ODS output 266
ODS DOCUMENT support 302, 318–319
BY statement, SGRENDER procedure 552

C

CALENDAR procedure 416
CALL DEFINE statement, COMPUTE statement (REPORT) 498–499, 570
caret (^) 148, 540
Cascading Style Sheet
See CSS
CAT= option, ODS DOCUMENT statement 303
CC= option, EMAIL engine 76
cell padding
changing style elements 386
differences in printer styles 394
reducing output file size 480
spacing inside tables 426–427
Cell style element 384–387
CELLPADDING= style attribute 386, 394, 426–427
cells, table
changing style attributes 496–497,
518–519, 534–535
changing style elements 384–387
CELLSPACING= style attribute 380–382, 422–425, 448
CELLSTYLE-AS statement, TEMPLATE procedure 282
CELLWIDTH= style attribute 508–509, 530
CENTER system option
adding images to output 108, 460, 464–465
controlling page margins 186
CFILL= option, PIE statement (GCHART) 556
CHART procedure
fixed-width fonts 401, 416
graphics output example 560–561
output objects and 206
CHARVAR generic definition 272
Chgbody template 378
CHISQ option, FREQ procedure 236

class selectors 357, 470, 476–477
CLASS statement, TABULATE procedure
controlling column widths 530
STYLE= option 514, 520
CLASS statement, TEMPLATE procedure
abstract style elements 370–371
color style elements 364
creating font definitions 414
modifying font size 402–403
modifying specific style elements 372
modifying typefaces 408–409
overview 362–363
setting fixed width 430
CLASSLEV statement, TABULATE procedure 520, 538
CLOSE action
ODS LISTING statement 224
ODS OUTPUT statement 216, 228
CODEBASE= option, ODS HTML statement 564
colon (:) 324
color
See also background color
See also foreground color
adding to color scheme 450–451
background images and 458
designing accessibility enhancements
452–453
identifying 444–445
modifying for hyperlinks 114
modifying in tables 448–449
modifying in titles 442–443
printing in 180–181
style definitions and 14, 357
style elements controlling 440–441
traffic lighting technique 282–283,
502–503
color_list style element 364, 394, 440–441
COLOR= option
FOOTNOTE statement 56
ODS PRINTER statement 180
TITLE statement 56
color picker tools 444–445
COLOR= style attribute 386
color style elements 364–365
COLORPRINTING system option 180,
190–191
colors style element 440, 450
column attributes 272–273
column headers
changing style attributes 494–495,
514–515, 532–533
spanning 278–279
column pointer control 262
COLUMN statement, TEMPLATE procedure 273, 278
column widths, controlling 508–509, 530–531
columns
calculating in custom templates 276–277
creating consistent printed pages 182–183
creating multiple 98–99, 184–185

formatting 264–265
generic 284–285
labeling results 264–265
COLUMNS= option
FILE PRINT ODS statement 284, 290
ODS PRINTER statement 184
ODS RTF statement 98
COLWIDTH= option, PROC REPORT statement 508–509
comma (,) 324
COMPRESS= option, ODS PDF statement 162–163
COMPUTE-AS statement, DEFINE statement (TEMPLATE) 276
COMPUTE statement, REPORT procedure
adding graphics to tables 504
adding traffic lighting 502
BEFORE_PAGE argument 570
CALL DEFINE statement 498–499, 570
IF statement 498–499, 502
LINE statement 504, 508–511
STYLE= option 504
Container element 357, 458
ContentFolder style element 377
ContentItem style element 377
ContentProcName style element 377, 428
CONTENTS= option
ODS HTML statement 32, 72
ODS PDF statement 156
ODS RTF statement 116, 118, 138
CONTENTS procedure 206, 238
Contents style element 377
ContentTitle style element 377
control strings
adding customized dates 104–105
aligning decimals 122–123
modifying table headers 144–145
rotating/aligning table headers 146–147
sending to RTF output 130–131
specifying metadata 104–105, 120–121, 484–485
style templates and 484–485
control words
for formatting 134–135
list of selected 128–129
COPY statement, DOCUMENT procedure 314
corporate style guidelines 470–471
CORR procedure
multiple output data sets 216
NOSIMPLE option 16, 236
output examples 14–18, 244
RTF output 214–215
crosstabulation table output 302
CSS (Cascading Style Sheet)
@media section 472–473
adding style property compatibility 386
class selectors 357, 470
creating alternate HTML formats 48–49
matching corporate style guidelines
470–471
multiple files with ODS HTML 474–475
SCREEN media section 472–473

simulating page breaks 38–41
style attribute names and 378
CSSSTYLE= option
@media style properties 472–473
ODS HTML statement 468–469
ODS PDF statement 468–469
ODS RTF statement 468–469
CSV output
importing ODS output to Excel 79
ODS MARKUP destination 322
overview 14, 18
CSV tagset 324–325, 338
curly braces {}
enclosing style attributes 494, 514, 532
in commands 321
special character protection and 126
custom table templates
calculating columns 276–277
creating custom styles 478–479
creating HTML 280–281
creating with DATA step 274–275
dynamic variables in 288–291
macro variables in 286–287
spanning column headers 278–279
traffic lighting technique 282–283
customizing
dates 104–105
headings for output 456–457
HTML output 56–57
metadata 164–165
ODS output 244–245
reports 271–291
style definitions 354–355
tagset templates 336–337
tagsets 321, 334–335

D

D3D style template 420–421
data cell-related style elements 384–387
DATA event 330
data set options 230–231
See also specific data set options
data sets
See output data sets
DATA step
assigning dynamic variables 290
ATTRIB statement 264
column pointer control 262
controlling output layout 262–263
creating custom reports 271–291
creating custom styles 478–479
creating custom table templates 274–275
creating markup files 321
DROP statement 260
ELSE DO group 266
FILE PRINT ODS statement 260–264, 270
FILE statement 76, 256–257
FORMAT statement 264, 284

formatting columns of results 264–265
inline formatting 268–269
KEEP statement 260
LABEL statement 264
labeling columns of results 264–265
labeling output object 258–259
LENGTH statement 268
naming output object 258–259
ODS OUTPUT CLOSE statement and 228
ODS output in 255–270
PUT _ODS_ statement 256, 260
PUT statement 256–257, 266–267, 333
selecting output variables 260–261
SET statement 268
table definitions 16, 272–273
Data style element 384–387, 534
DataEmphasis style element 400
DATE field (Microsoft Word) 126
DATE system option 102
dates
customized 104–105
dynamic 102–105
static 102–105
decimals, aligning in RTF output 122–123
Default style template
accessibility considerations 452
adding background images 458
basing templates on 352
changing spacing inside tables 426–427
fixed-width fonts 416
font elements 366
formatting footnotes 390
formatting titles 390
graphics procedures 28
identifying colors 444–445
identifying font definitions 400–401
inheritance and 358–361, 370–371, 380, 384
modifying color in titles 442
modifying default font sizes 402–407
modifying default typefaces 408
modifying style elements 384–387
modifying table borders 420–421
overview 346, 358–363
printer styles and 394–395
printing in color 180
removing table borders/rules 422–425
setting width for HTML output 430–431
DEFAULT tagset template 328
DefaultNewTitle style template 354
DefaultSmaller style template 402
DefaultSqueeze style template 426
DEFINE EVENT statement, TEMPLATE procedure 333
DEFINE HEADER statement, TEMPLATE procedure 272, 278
DEFINE statement, REPORT procedure
GROUP option 512
indenting values 540–541
ORDER option 512
STYLE= option 122, 158, 496–499

WIDTH= option 508–509
DEFINE statement, TEMPLATE procedure
COMPUTE-AS statement 276
creating graphics output 552–553
FORMAT= option 274
HEADER= option 274
modifying default font sizes 402–403
overview 350–353
WIDTH= option 430
DEFINE STYLE statement, TEMPLATE procedure 430
DEFINE TAGSET statement, TEMPLATE procedure 333
DELETE option, OBPAGE statement (DOCUMENT) 310
DELIMITER= option, ODS CSV statement 324
DEST function, ODS ESCAPECHAR statement 58–59
device drivers
documentation on 558
generating internal graphs 303
graphics output and 28, 554, 558–561
Zdevices and 545
DEVICE= option, GOPTIONS statement 554, 560–561, 568
$DFONT format 282
diagnostic tagsets 330–331, 358, 364
DICTIONARY.STYLES file 490
DIR statement, DOCUMENT procedure
312–313
<DIV> tag 26–27
%DO statement 226
DOC event 330, 334
DOC_HEAD event 332–333
DOC_SEQNO= data set option 318
DOC= suboption
ODS CSV statement 324, 340–341
ODS TAGSETS.RTF statement 140
overview 340–341
DocBook destination 322
DOCBOOK format 322–323
DOCBOOK tagset template 322–323, 328, 332–333
docFont attribute 401
DOCUMENT destination
modifying page breaks 310–311
modifying titles 310–311
output example 304–309
overview 293–294, 302–303
Document element 357
DOCUMENT procedure
COPY statement 314
copying objects into document stores
314–315
creating document stores 308, 312–313
DIR statement 312–313
IMPORT statement 320
LIST statement 310
MAKE statement 312–313
managing document stores 298–299
manipulating page breaks 316–317
manipulating titles 316–317
modifying document stores 312–313
modifying page breaks 310–311

modifying titles 310–311
NAME= option 312–313
NOTE statement 320
OBANOTE statement 320
OBBNOTE statement 320
DOCUMENT procedure (*continued*)
OBFOOTN statement 320
OBPAGE statement 310–311, 320
OBSTITLE statement 310–311, 320
OBTEMPL statement 303
OBTITLE statement 320
OBTITLE1 statement 310
overview 293–294
replaying document stores 300–301
run-group processing 298
SETLABEL statement 312–313
document stores
characteristics 302
copying objects into 314–315
creating 294
creating interactively 304–309
creating with DOCUMENT procedure 308, 312–313
defined 294
defining folder hierarchy 304–309
managing 298–299
modifying organization 312–313
replaying 300–301, 303
SAS/GRAPH output 303
SASEDOC LIBNAME engine 318–319
templates and 303
user-defined formats and 302
viewing 294
documentation
changing style attributes 532
displaying tagset information 324
frame navigation 376–377
on device drivers 558
procedures supporting ODS graphics 546
running legacy code 560
style elements 371, 476
table style elements 382
Zdevices 545
Documents window
creating document stores 304–309
overview 296–297
viewing document stores 294
DPI (dots-per-inch) 558
drill-downs in output tables 66–67
DROP= option, ODS OUTPUT statement 230
DROP statement 260
dynamic dates, adding 102–105
DYNAMIC statement, TEMPLATE procedure 289
DYNAMIC= suboption, FILE PRINT ODS statement 284, 290
dynamic variables
as attribute values in templates 288–291
assigning text 284
supplying values to attributes 289

E

e-mail, sending HTML results 76–77
ELSE DO group 266
ELSE statement 282, 502
EMAIL engine
ATTACH= option 76
CC= option 76
FROM= option 76
overview 76–77
SUBJECT= option 76
TO= option 76
EmphasisFont attribute 400–401
END= attribute 278
escape characters
See ODS ESCAPECHAR statement
EVENT_MAP tagset 330
events, modifying 332–333
Excel Spreadsheet Markup Language 14, 18, 78–79
EXCELXP destination 322
EXCELXP tagset 78, 328, 338–339
EXCLUDE action, ODS RTF statement 244, 248
EXCLUDE option, ODS PRINTER statement 189
excluding output objects 208, 240–241
EXIST macro function 490
EXPORT procedure 79
EXPORT Wizard 79
ExtendedPage style element 400
Extensible Markup Language (XML) 321
external graphs 303
external image files 558–559

F

FancyPrinter style template 346
field codes
inserting into RTF output 126–129
sending only to RTF output 130–131
style templates and 484–485
FILE= option
ODS DOCUMENT statement 294
ODS HTML statement 27
ODS PDF statement 152
ODS PS statement 192
ODS TAGSETS.ODSSTYLE statement 364
FILE PRINT ODS statement
caveats for using 270
COLUMNS= option 284, 290
controlling output layout 262–263
creating custom styles 479
creating output 256–257
DYNAMIC= suboption 284, 290
FOOTNOTES option 270
FORMAT= option 290
formatting columns of results 264–265
GENERIC= suboption 284
LABEL= option 290
labeling columns of results 264–265
LINESIZE= system option 270

NOFOOTNOTES option 270
NOTITLES option 270
OBJECT= suboption 258–259
OBJECTLABEL= suboption 258–259
output options 270
PAGESIZE= system option 270
selecting output variables 260–261
table definitions 272
TEMPLATE= suboption 274, 290
TITLES option 270
traffic lighting technique 282
VARIABLES= suboption 260–262, 264
FILE statement 76, 256–257
FILENAME statement
EMAIL engine 76
MOD option 70, 72
overview 70, 72
fixed-width fonts 401, 408, 416–417
FixedEmphasisFont attribute 401
FixedFont attribute 400–401
FixedStrongFont attribute 401
FLYOVER= option, ODS ESCAPECHAR statement 158, 160
FMTLIB option, PROC FORMAT statement 208
FONT= attribute 450
font attributes 358, 450
font definitions
creating 414–415
identifying 400–401
modifying default typefaces 408–409
modifying font sizes 402–407
modifying font styles 412–413
modifying font weights 412–413
modifying font widths 412–413
FONT_FACE= style attribute 282, 358, 366
font families 410, 452
FONT= option
FOOTNOTE statement 56
TITLE statement 56
FONT_SIZE= style attribute 358, 366
font sizes
controlling output with 480
modifying for HTML output 402–403
modifying for Printer output 404–405
modifying for RTF output 406–407
font style elements 366–367
FONT_STYLE= style attribute 358, 366
font styles, modifying 412–413
<FONT> tag 56–57, 60–61
font typeface 408–409, 494
FONT_WEIGHT= style attribute 358, 366
font weights, modifying 412–413
font widths, modifying 412–413
FONTREG procedure 410
FONTRENDERING= system option 560
fonts
accessibility considerations 452
differences in RTF style 396
fixed-width 401, 408, 416–417

fixing in PowerPoint 132
identifying available 410–411
in LISTING destination 256
inline formatting 268–269
modifying for output 56–57
modifying for procedures 416–417
modifying typefaces 408–409
style definitions and 14, 16, 357
traffic lighting technique 282–283
fonts style element 400–405, 407
FooterEmphasis style element 400
footers, Microsoft Word support 86–87
FOOTNOTE statement
adding images to output 462
BOLD option 56
COLOR= option 56
control words 128–129, 135
field codes 126–129
FONT= option 56
HEIGHT= option 56
HTML tags in 56–57
inline formatting 60–61
ITALIC option 56
JUST= option 56
JUSTIFY=RIGHT option 62
LINK= option 56
FOOTNOTE statement (*continued*)
macro variables in 62
RTF control strings 484
script support 62
style elements and 392
FOOTNOTEFONT= attribute 414
footnotes
See also FOOTNOTE statement
adding annotations 160
modifying color of 442
modifying using style elements 390–393
placing text in document 106–107
FOOTNOTES option, FILE PRINT ODS statement 270
foreground color
modifying 442–443
style definitions 364–365
FOREGROUND= style attribute
adding colors to color scheme 450
adding property compatibility 386
traffic lighting technique 282, 502, 524
FOREGROUNDCOLOR= style attribute 358
FORMAT= option
DEFINE statement 274
FILE PRINT ODS statement 290
FORMAT procedure
FMTLIB option 208
indenting values 540–541
overview 208–209
row style attributes 520–521
VALUE statement 540–541
FORMAT statement 264, 284
formats

HTML supported 326
user-defined 302, 503
formatting
changing for BOX= option 516–517
columns of results 264–265
control words for 134–135
footnotes with style elements 390
inline 60–61, 268–269
markup process 321
titles with style elements 390
traffic lighting technique and 282
FORMCHAR= system option 416
FORMDLIM= system option 4
forward slash (/) 321, 496
frame navigation 32–35, 376–377
FRAME= option, ODS HTML statement 32, 72
FRAME= style attribute 382, 422–425
Frame style element 377
<FRAME> tag 34–35
<FRAMESET> tag 34–35
FREQ procedure
CHISQ option 236
creating output data sets 216
document stores and 308, 314–315
handling page breaks 88–89
table templates and 302
FROM= option, EMAIL engine 76

G

G3D procedure 568
$GALFT format 160
GAREABAR procedure 568
GBARLINE procedure 563, 568
GCHART procedure
graphics output and 558–559, 568
HTML output and 554
output objects and 206
PIE statement 556, 562
style template support 544
VBAR statement 28
GCONTOUR procedure 563, 568
GENERIC column attribute 272–273, 288
generic columns in table templates 284–285, 288
GENERIC= suboption, FILE PRINT ODS statement 284
GETDATE program 374
GhostScript 193
GIF format 28, 454, 545
GLM procedure
modifying typefaces 408–409
output data sets and 223
output objects and 200–202
GMAP procedure 544, 568
GOPTIONS statement
changing output via options 556–557
creating graphic images 548
DEVICE= option 554, 560–561, 568
HSIZE option 559
overview 544–545

specifying device drivers 554
VSIZE option 559
XMAX option 559
XPIXELS option 559, 566
YMAX option 559
YPIXELS option 559, 566
GOUT= option
ODS DOCUMENT statement 303
ODS HTML statement 570
GPATH option, ODS HTML statement 28–29
GPLOT procedure 544, 568
GRADAR procedure 568
graph template definitions
creating output 552–553
overview 16, 544–545
removing access 488
storing 357
Graph Template Language (GTL) 544, 552–553
GraphFonts element 366
graphics output
adding to tables 504–505, 526–527
changing via SAS/GRAPH options 556–557
creating with graph templates 552–553
creating with ODS GRAPHICS statement 546–547
creating with ODS Statistical Graphics 548–549
creating with SAS/GRAPH 554–555
creating with SG procedures 550–551
creating with SGRENDER procedure 552–553
devices and destinations 558–561
HTML support 28–29, 562–567
overview 544–545
RTF support 568–569
text support 570–571
GraphTitleText element 556
GRESEG catalogs 303
GROUP option, DEFINE statement (REPORT) 512
GSTYLE system option 544–545, 554
GTILE procedure 568
GTITLE option, ODS HTML statement 556
GTL (Graph Template Language) 544, 552–553

H

<H1> tag 56–57
<HEAD> tag
adding JavaScript 62
CSS files and 474
customized metadata 164
overview 26–27
HEADER event 330
HEADER= option, DEFINE statement 274
header-related style elements 384–387
Header style element 370–373, 384–387
HeaderEmphasis style element 400
headers

adding dates 102–103
bookmark comparison 154
customizing for output 456–457
default area for titles 108
differences in RTF style 396
Microsoft Word support 86–87
modifying default font sizes 402–403
modifying for tables 144–145
modifying style attributes 494–495,
514–515
modifying style elements 384–387
rotating/aligning 146–147
spanning 278–279
underlining 289
HeadersAndFooters style element
identifying font definitions 400
inheritance and 370–371, 384
modifying colors in tables 448
headingFont attribute 401
HEADTEXT= option, ODS HTML statement 62, 120
HEIGHT= option
FOOTNOTE statement 56
TITLE statement 56
HighContrast style template 452–453
histograms 550
hit counters 62
hot spots 562–563
HR_MACRO table template 286
HR_SPAN table template 278
<HR> tag 368
HSIZE option, GOPTIONS statement 559
HTML (Hypertext Markup Language) 321, 558
HTML class attribute 476
HTML destination output
accessibility considerations 452
adding background images 458–459
adding custom headings 456–457
adding images to style elements 388–389
adding JavaScript 62–63
adding logo to output 454, 456–457
adding scripts 62–63
appending to existing file 70–71
appending to existing table of contents
72–73
applying CSS file 468–469
applying inline formatting 60–61
as output destination 14
breaking up into files 36–37
HTML destination output (*continued*)
color considerations 444–445
combining procedure results 30–31
comparing listing output 434–435
controlling page breaks 94–95
controlling page orientation 91
creating alternate formats 48–49
creating custom table definitions 280–281
creating graphics output 562–567
creating multiple columns 98

creating multiple copies 50–51
creating with just results 68–69
creating with ODS MARKUP statement 326–327
creating with SAS/GRAPH 554–555
customizing 56–57
DATA step processing 256–257
drill-downs in output tables 66–67
examples 8, 15, 18–19
external image files 558–559
fixed-width fonts 416
formats supported 326
frame navigation and 32–35, 376–377
generating 24–25
generating table of contents 32–33
graphics output 28–29, 548–549, 558–561
hyperlinks and anchors 52–55
identifying available fonts 410–411
importing into spreadsheet 78–79
indenting values 540–541
inserting images with style templates 482
justification style attribute 268
limiting tags in 58–59
modifying background colors 446–447
modifying font sizes 402–403
modifying procedure labels 42–45
modifying table borders 420–421
modifying title bar 46–47
modifying typefaces 408
moving into Internet/intranet site 74–75
numbers in table of contents 428–429
ODS CLOSE statement and 228
ODS MARKUP destination 322
overview 26–27
page breaks 30
PDF output comparison 394–395
reducing output file size 480
removing table borders/rules 422–423
replaying document stores 300–301
SELECT statement and 244
sending results via e-mail 76–77
setting width 430–431
simulating page breaks with CSS 38–39
spacing inside tables 426
special characters 321
style definitions 14, 346, 357, 368
HTML= option, PIE statement (GCHART) 562
HTML style element 368–369
<HTML> tag 26–27
HTML tags 280, 321
See also specific tags
HTML tagsets 326
HTMLCLASS= style attribute 280, 358–359, 476
HTMLSTYLE= attribute 476
$HWYRT format 264
hyperlinks
adding to RTF output 114–115, 148–149
drill-downs in output tables 66–67
in PDF output 158–161

linking results to Web sites 54–55
modifying text color 114
overview 52–53
Hypertext Markup Language (HTML) 321, 558

I

<I> tag 56–57
ID= option
ODS HTML statement 80
ODS PRINTER statement 188
IDX anchor tag names 54
IF statement
CELLSTYLE-AS statement and 282
COMPUTE statement, REPORT procedure 498–499, 502
images
adding logo to output 454–457
adding to each page of output 462–463
adding to printer output 464–465
adding to RTF output 108–109, 460–461
adding to style elements 388–389
background 458–459
creating graphics 548–549
creating with GOPTIONS statement 544
external image files 558–559
inserting with style templates 482–483
<IMG> tag
adding logo to output 456
adding to style elements 388–389
creating graphic images 548
graphics output example 558–559
HTML output with SAS/GRAPH 554
overview 28
IMPORT statement, DOCUMENT procedure 320
importing
difficulties with listing output 6–7
files into word processors 84–85
ODS output into spreadsheet 78–79
RTF output into slide show 132–133
indexes, and style elements 376
IndexProcName style element 428
IndexTitle style element 400
inheritance, and style elements 358–361, 370–371, 380, 384
inline formatting 60–61, 268–269
Internet/intranet site, moving ODS output into 74–75
ITALIC option
FOOTNOTE statement 56
TITLE statement 56
item stores 273, 294

J

Java language
creating HTML output 564–565
frame navigation 32
generating internal graphs 303
graphics output and 544, 560–561
special output effects 62

JAVAIMG device driver 303
JavaScript language
frame navigation 32, 377
pop-up windows 374
special output effects 62
Journal style template
adding hyperlinks to output 146
changing definitions 348
creating RTF files 82
enhancing accessibility 452–453
JPEG format
adding images to output 108, 460
adding logo to output 454
graphics procedures 28
JUST= option
FOOTNOTE statement 56
ODS ESCAPECHAR statement 108
TITLE statement 56
JUST= style attribute
adding images to output 460
decimal justification 122
in table definitions 272, 288–291
inline formatting and 268
resetting 392
justification style attribute
decimal justification 122
HTML output 268
PDF output 268
RTF output 122, 268
table templates 273, 288–291
JUSTIFY=RIGHT option
FOOTNOTE statement 62
TITLE statement 104
JUSTIFY= table attribute 273, 288–291

K

KEEP= data set option 260–261
KEEP= option, ODS OUTPUT statement 230
KEEP statement 260

L

LABEL option, ODS TRACE statement 202
LABEL= option
BOX= option, TABLE statement (TABULATE) 516
FILE PRINT ODS statement 290
LABEL statement 264
PRINT procedure 56–57
labels
See also LABEL statement
displaying in bookmarks 154
for columns of results 264–265
hiding procedure titles 436
modifying 42–45
output objects 258–259
removing 16, 428
selecting output objects by 213
specifying as quoted strings 204

LaTeX output
graphics output and 548–549
ODS MARKUP destinations and 322
overview 14, 18
LAYOUT GRIDDED statement 552
LAYOUT OVERLAY statement 552
LEFT attribute 510
LEFTMARGIN= style attribute 378, 538–541
LEFTMARGIN= system option 96, 186, 432
LENGTH statement 268
%LET statement
adding customized dates 104–105
assigning macro variable values 287, 332
defining macro variables 287
specifying alternate templates 490
LIBNAME engines 318–319, 321
LIBNAME statement 318–319, 486
line breaks, RTF output 104–105
line plots 550
LINE statement, COMPUTE statement (REPORT) 504, 508–511
LINESIZE= system option 91, 270
LINK= option
FOOTNOTE statement 56
TITLE statement 40, 56
<LINK> tag 474
LIST statement
DOCUMENT procedure 310
TEMPLATE procedure 354
LIST STYLES statement, TEMPLATE procedure 346, 354
LISTING destination output
closing 226
comparing output 434–435
controlling column widths 530
DATA step processing 256–257
difficulties importing 6–7
FILE PRINT ODS statement 270
graphics output and 545–549, 554, 558
inserting blank lines 510–511
limitations 4–5
ODS output comparison 9
output data sets 506
output examples 18–19
procedures with ODS output 208
replaying document stores 300–301
sending control strings 130
sending field codes 130
simulating 434–435
style template support 544–545
underlining headers 289
LISTING option, ODS TRACE statement 200
LISTING window 546–547
LOGISTIC procedure 220–221, 224–225
logos, adding to output 108, 454–457
logs
See SAS logs
LOOKLIST style 434–435

M

macro programs, drill-downs in output tables 66–67
macro variables
assigning values 104, 286, 289, 332
automatic 287
creating 490
declaring 286
defining 62, 286
in custom table definitions 286–287
inside template definitions 333
MAKE statement, DOCUMENT procedure 312–313
MARGINLEFT= style attribute 378, 538–541
MARGINRIGHT= style attribute 378
markup concept 321–323
MARKUP destinations
diagnostic tagsets 330–331, 358
HTML support 326
overview 321–323
ways to use 322
MATCH_ALL option, ODS OUTPUT statement 222
MEANS procedure
document stores and 308, 314–315, 318–319
ODS output and 208
output data sets 528
@media section (CSS) 472–473
META tag 68–69
metadata
customized for PDF output 164–165
specifying in RTF output 120–121
METATEXT= option, ODS HTML statement 120
Microsoft Excel 79, 568
Microsoft PowerPoint 132–133, 568
Microsoft Word
completing table of contents 118–119
difficulties importing listing output 6–7
field codes 126–127
graphics output and 568
importing files 84–85
RTF output 124–125
viewing properties 120
working with headers and footers 86–87
Minimal style template 50–51, 346, 458
missing values in output objects 222
MOD option, FILENAME statement 70, 72
MonochromePrinter style template 452–453
MSOFFICE2K destination 322
multi-page reports 100–101
multiple columns, creating 98–99, 184–185
multiple output data sets, creating 216–217
multiple output objects, selecting 238–239
multiple procedure labels, changing 44–45
multiple procedures
creating output data sets from 220–223
selecting output for 242–243
MVAR statement, TEMPLATE procedure 286–287, 332

N

NAME= option

ODS DOCUMENT statement 294
PROC DOCUMENT statement 312–313
NAMEDHTML tagset 330
naming
anchor tags 52–55
groups of style attributes 357
output objects 258–259
navigation
frame 32–35, 376–377
through document stores 314
  (HTML) 538
NEWFILE=BYGROUP option, ODS HTML statement 36, 40
NEWFILE= option, ODS HTML statement 30, 40
NEWFILE=PAGE option, ODS HTML statement 36
NEWFILE=PROC option, ODS HTML statement 36, 40
NMVAR statement, TEMPLATE procedure 286
NO_BOTTOM_MATTER suboption, ODS HTML statement 68–73, 80
NO_TOP_MATTER suboption, ODS HTML statement 68, 70, 80
NOCENTER system option
adding images to output 464–465
controlling page margins 186
inline formatting 60–61
modifying style attributes 378
NOCOLOR option, ODS PRINTER statement 180
NOCOLORPRINTING system option 180
NOFOOTNOTES option, FILE PRINT ODS statement 270
NOGSTYLE system option 544–545, 554, 560
NOGTITLE option, ODS HTML statement 556
NONE argument, ODS EXCLUDE statement 242, 248
NONUMBER system option 100
NOPDFNOTE option, ODS PDF statement 158
NOPRINT procedure option 224
NOQUOTELENMAX system option 112
NOSIMPLE option, CORR procedure 16, 236
NOTE statement, DOCUMENT procedure 320
Notepad, RTF output 124–125
NOTITLES option, FILE PRINT ODS statement 270
NOTOC_DATA option, ODS RTF statement 118
NOTOC option, ODS PDF statement 166
NUMBER system option 100
numeric variables, generic definitions 272, 284–285
NUMPAGES field (Microsoft Word) 126
NUMVAR generic definition 272

O

OBANOTE statement, DOCUMENT procedure 320
OBBNOTE statement, DOCUMENT procedure 320
OBFOOTN statement, DOCUMENT procedure 320
OBJECT= suboption, FILE PRINT ODS statement 258–259
<OBJECT> tag 564
OBJECTLABEL= suboption, FILE PRINT ODS statement 258–259
OBPAGE statement, DOCUMENT procedure 310–311, 320
OBSTITLE statement, DOCUMENT procedure 310–311, 320
OBTEMPL statement, DOCUMENT procedure 303
OBTITLE statement, DOCUMENT procedure 320
OBTITLE1 statement, DOCUMENT procedure 310
ODS (Output Delivery System)
output destinations 18–19

output objects 18–19
overview 14–15
references and resources 20
style definitions 16–17
table definitions 16–17
ODS ALL_CLOSE statement 136, 188
ODS CHTML statement 48–49, 326, 480
ODS CSV statement
DELIMITER= option 324
DOC= option 324, 340–341
importing output to Excel 79
overview 324
ODS CSVALL statement 324
ODS DOCUMENT statement
CAT= option 303
FILE= option 294
GOUT= option 303
identifying object names 202
NAME= option 294
overview 294–295
WRITE option 294
ODS document stores
See document stores
ODS documents
defined 293–294
replaying 300–301, 310–311
ODS Documents window
See Documents window
ODS ESCAPECHAR statement
adding hyperlink to output 148
applying inline formatting 60–61
defining escape characters 168
DEST function 58–59
FLYOVER= option 158, 160
indenting values 538, 540–541
inserting field codes into RTF output
126–127, 129
inserting logos 460
JUST= option 108
modifying titles and footnotes 390
overview 58–59
page numbering 100
PREIMAGE= attribute 108
RAW method 134
RTF control strings 484
STARTPAGE=OFF option 110
STYLE function 60–61, 160
SUPER function 60–61
URL= option 114, 158
ODS EXCLUDE statement
complex combinations of output objects 246
identifying output object names 202
limiting output objects 546
NONE argument 242, 248–249
ODS SELECT statement and 240–241
overview 240–241
PERSIST option 242, 248–249
specifying labels 204

ODS GRAPHICS statement
altering defaults 548
generating graphics output 546–547
OFF argument 544
ON argument 544, 548
ODS HTML statement
ANCHOR= option 562
BODY= option 27, 30, 32
class selectors 476–477
CLOSE action 28
CODEBASE= option 564
CONTENTS= option 32, 72
creating alternate outputs 48–49
creating multiple copies of same output
50–51
creating output with SAS/GRAPH
554–555
CSSSTYLE= option 468–469
FILE= option 27
FRAME= option 32, 72
GOUT= option 570
GPATH option 28–29
graphics output 28, 546–547, 554, 568
GTITLE option 556
HEADTEXT= option 62, 120
ID= suboption 80
indenting values 538
METATEXT= option 120
multiple CSS files 474–475
NEWFILE= option 30, 40
NEWFILE=BYGROUP option 36, 40
NEWFILE=PAGE option 36
NEWFILE=PROC option 36, 40
NO_BOTTOM_MATTER suboption
68–73, 80
NOGTITLE option 556
NO_TOP_MATTER suboption 68, 70, 80
overview 326
PAGE= option 36
PATH= option 28–29, 546, 554, 557
SELECT action 244, 250
style definitions 346
STYLE= option 490
URL= option 554
ODS HTML3 statement 326
ODS HTMLCSS statement 326
ODS LISTING statement 224, 506
ODS MARKUP destinations
See MARKUP destinations
ODS MARKUP statement
creating CSV files 324–325
creating HTML result files 326–327
creating XML files 328–329
TAGSET= option 322
tagset template definitions 16, 273
TAGSETS.RTF destination 136
TYPE= option 322
ODS MSOFFICE2K statement 78–79, 326

ODS NOPROCTITLE statement 436–437, 460
ODS NOPTITLE statement 42, 368
ODS option, PUT statement 256, 260, 272
ODS output
See also output data sets
See also output objects
complex combinations of output objects 246–247
controlling layout 262–263
creating PDF files 192–193
customizing selections 244–245
default style templates 490
excluding 240–241
exploring 197–209
FILE PRINT ODS statement 270
formatting columns of results 264–265
identifying exclusions 248–249
identifying output objects 198–203
identifying selections 248–249
importing into spreadsheet 78–79
in DATA step 255–270
inline formatting 268–269
labeling columns of results 264–265
limiting 235–253
modifying with PUT statement 266–267
moving into Internet/intranet site 74–75
overview 14, 18–19
procedure examples 206–209
selecting for multiple procedures 242–243
selecting multiple output objects 238–239
selecting results subset 236–237
selecting WHERE conditions 250–253
sending directly to printer 172–173
style definitions 346
varieties in formats 8–11
viewing results 204–205
ODS OUTPUT statement
CLOSE action 216, 228
creating multiple output data sets 216–217
creating single output data sets 212–213, 232
DROP= option 230
identifying output object names 202
KEEP= option 230
MATCH_ALL option 222
PERSIST option 222–223
RENAME= option 230
specifying labels 204
ODS PATH statement
APPEND option 487
finding style definitions 354
PREPEND option 486–488
READ option 486
REMOVE option 488
removing access to templates 488–489
RESET option 488
retrieving templates 486–487
SHOW option 487–488
simulating listing output 434–435
specifying alternate templates 490

storing templates 486–487
tagset template definitions 273, 338
UPDATE option 486
ODS PCL statement 176
ODS PDF statement
ANCHOR= option 166
BOOKMARKGEN= option 156
BOOKMARKGEN=NO option 156
BOOKMARKGEN=YES option 156
BOOKMARKLIST= option 156
BOOKMARKLIST=HIDE option 156
BOOKMARKLIST=NONE option 156
BOOKMARKLIST=SHOW option 156
COMPRESS= option 162–163
CONTENTS= option 156
CONTENTS=YES option 156
creating multiple copies of same output
50–51
CSSSTYLE= option 468–469
FILE= option 152
indenting values 538
inserting images with style templates 482
NOPDFNOTE option 158
NOTOC option 166
PDFNOTE= option 158
PDFNOTE=NO option 158
PDFTOC= option 156
STARTPAGE= option 166
STARTPAGE=NO option 170
STARTPAGE=NOW option 170
system options affecting 190–191
UNIFORM option 182
ODS PHTML statement 48–49, 326, 480
ODS POSTSCRIPT statement 163
ODS PRINTER statement
adding images to output 464
CLOSE action 176
COLOR= option 180
COLOR=NO option 180
ODS PRINTER statement (*continued*)
COLUMNS= option 184
EXCLUDE option 189
ID= option 188
multiple versions of same output 188–189
NOCOLOR option 180
overview 172–173
PDF option 163
PDFMARK option 165, 193
PRINTER= option 174, 192
PRINTER=HOST option 178
PRINTER=POSTSCRIPT option 178
PS option 176, 179, 410
SAS option 178, 410
SELECT option 189
STARTPAGE= option 464–465
system options affecting 190–191
UNIFORM option 182
ODS PROCLABEL statement

hiding procedure titles 436
labels displaying in bookmarks 154
modifying procedure labels 42, 44
modifying procedure titles 116
naming output objects 258
overriding style element values 368
removing text labels 428
TITLE2 statement 42
ODS PS statement
FILE= option 192
PDFMARK option 165
ODS PTITLE statement 436
ODS RESULTS statement 226
ODS RTF statement
AUTHOR= option 120, 124
BODYTITLE option 86, 100, 106, 108
BODYTITLE_AUX option 106
COLUMNS= option 98
CONTENTS= option 116, 118, 138
creating graphs as ActiveX 568–569
creating multiple copies of same output
50–51
creating RTF files 82–83
CSSSTYLE= option 468–469
customized metadata in 164
EXCLUDE action 244, 248
indenting values 538
NOTOC_DATA option 118
OPERATOR= option 120, 124
PREPAGE= option 138–139
RTF control strings 484
SASDATE option 102
SELECT action 250
SHOW action 248
STARTPAGE= option 94
STARTPAGE=NO option 94, 242
STYLE= option 432
TEXT= option 110, 114, 130
TITLE= option 120, 124
TOC_DATA option 116, 118
WORDSTYLE option 138
ODS SELECT statement
ALL option 242, 249
complex combinations of output objects 246
identifying output object names 202
limiting output objects 546
multiple output objects 238
ODS EXCLUDE statement and 240–241
PERSIST option 242
selecting result subsets 236
specifying labels 204
ODS SHOW statement 248–249
ODS Statistical Graphics framework
See also graphics output
automatic graph creation 550–551
creating graphic images 548–549
Graph Template Language 544, 552–553
output objects and 206

overview 543
template definitions 16
ODS TAGSETS.CSVBYLINE statement 324
ODS TAGSETS.ODSSTYLE statement 364
ODS TAGSETS.RTF statement
adding hyperlinks to output 148–149
CLOSE action 136
DOC= suboption 140
generating table of contents 142–143
modifying table headers 144–145
PREPAGE= option 138–139
suboptions list 140–141
TRHDR suboption 144, 146
TROWD suboption 144, 146
TROWHDRCELL suboption 144, 146
ODS TAGSETS.<*tagsetname*> statement 322
ODS TAGSETS.XHTML statement 326
ODS TEXT= statement 110–113, 166
ODS TRACE statement
complex combinations of output objects 246
excluded output objects and 208
identifying output objects 198–201, 238
LABEL option 202
LISTING option 200
ON argument 212, 232, 552
table templates 271
viewing output objects 204–205
ODSSTYLE tagset 330, 364
OFF argument
ODS GRAPHICS statement 544
ODS RESULTS statement 226
ON argument
ODS GRAPHICS statement 544, 548
ODS RESULTS statement 226
ODS TRACE statement 212, 552
OPEN macro function 490
OpenOffice, importing files 84–85
OpenVMS operating system 581
operating system differences 575–582
OPERATOR= option, ODS RTF statement 120, 124
OPTIONS statement
See specific system options
ORDER option, DEFINE statement (REPORT) 512
ORDER_DATA table attribute 273
ORIENTATION= system option
controlling column widths 530
controlling page margins 186
creating multiple columns 184–185
impacting output 90–91, 190–191
OUT= option
PROC REPORT statement 232, 506
PROC TABULATE statement 528
output data sets
as output destination 14
closing 228–229
combining results 218, 226–227
comparing to printed output 214–215

creating 11, 212–213
creating from multiple procedures 220–223
creating from procedures 218–219, 232–233
creating from REPORT procedure 232–233, 506–507
creating from TABULATE procedure 528–529
creating multiple 216–217
data set options 230–231
examples 18–19
opening 228–229
selecting subset 236–237
without printed output 224–225
Output Delivery System
See ODS
output objects
See also output data sets
comparing to printed output 214–215
copying into document stores 314–315
creating 271
diagnostic tagsets 330–331, 358
excluded 208, 240–241
finding names 225
identifying 198–203
labeling 258–259
missing values in 222
naming 258–259
overview 16, 18–19
page breaks after copying 316–317
procedure examples 206–209
selecting by labels 213
selecting complex combinations 246–247
selecting multiple 238–239
storing information about 303
titles after copying 316–317
viewing 204–205
OUTPUT option, ODS statement 202, 204
Output style element 382, 420–425
output tables, creating drill-downs 66–67
OUTPUTWIDTH= attribute 430, 530

P

page breaks
after copying output objects 316–317
for DOCUMENT destination 310–311
graphics output and 570
<HR> tag and 368
in HTML output 30
in PDF output 40, 168–171
in RTF output 88–89, 94–95
NEWFILE= option, ODS HTML statement 40
simulating with CSS 38–39
TAGSETS.RTF destination 136
PAGE field (Microsoft Word) 126
page margins
in PDF output 186–187
in RTF output 96–97, 432–433

indenting values 538–541
TAGSETS.RTF destination 136
page numbering
in PDF output 168–169
in RTF output 100–101, 130–131
PAGE= option, ODS HTML statement 36
page orientation, controlling 90–91
page size, controlling 92–93, 186–187
PAGEBREAKHTML= style attribute 378
PageBreakLine style element 368
PAGESIZE= system option 91–92, 186, 270
PANELBY statement, SGPANEL procedure 550
PAPERSIZE= system option 186
PARENT= statement, TEMPLATE procedure
adding images to printer output 464
custom tagsets 333–335
modifying default font sizes 402–404
modifying tagset events 332
style template definitions 352
PARENT style definition 408, 412
parentheses () 364
PATH= option, ODS HTML statement 28–29, 546, 554, 557
PCL (Printer Control Language)
as output destination 14, 394
creating files 176
graphics output and 558
ODS output 10
PDF destination output
adding images to style elements 388–389
adding logo to output 454
annotations 158–161
applying CSS file 468–469
comparing listing output 434–435
compression levels 162–163
controlling column widths 530
converting with Adobe 192–193
creating bookmarks in PDF file 154–155
creating consistent printed pages 182–183
creating linked set of results 166–167
creating multiple columns 184–185
creating multiple versions 188–189
creating PDF files 152–153
creating PostScript files 176–177
DATA step processing 256–257
examples 9–10, 18–19
external image files 558
graphics output 548, 558–561
HMTL output comparison 394–395
hyperlinks 158–161
indenting values 540–541
inserting images with style templates 482
justification style attribute 268
modifying bookmark display 156–157
optimizing size 162–163
overview 14, 151, 394
page breaks in 40, 168–171
page margins in 186–187
page numbering in 168–169

printer drivers 178–179
printing in color 180–181
reducing output file size 480
replaying document stores 300–301
RTF output comparison 396
selecting printers 174–175
sending directly to printer 172–173
system options affecting 190–191
table of contents 42
with customized metadata 164–165
PDF option, ODS PRINTER statement 163
PDFC device driver 558
PDFMARK option
ODS PRINTER statement 165, 193
ODS PS statement 165
PDFNOTE= option, ODS PDF statement 158
PDFTOC= option, ODS PDF statement 156
PDV (Program Data Vector) 256, 260, 262
period (.) 321
Perl language, special output effects 62
PERSIST option
ODS EXCLUDE statement 242, 248–249
ODS OUTPUT statement 222–223
ODS SELECT statement 242
PHP language 32, 377
pie charts 556–557, 562–563
PIE statement, GCHART procedure
CFILL= option 556
HTML= option 562
PLOT procedure 401, 416
PMML tagset 328
PNG format
adding images to output 108, 460
adding logo to output 454
graphics output and 28, 554, 558–561
ODS Graphics support 548
pop-up menus 300–301
pop-up notes 158
pop-up windows 358–359, 374
POSTHTML= style attribute 280, 374, 456
POSTIMAGE= style attribute 388, 482–485
PostScript files
adding logo to output 454
as output destination 14, 394
creating from output 176–177
graphics output and 558
ODS output 10
overview 178
pound sign (#) 274, 318
PREFIX1= style attribute 368
PREHTML= style attribute 280, 374, 456–459
PREIMAGE= style attribute
adding graphics to tables 504, 526
adding images to output 460
adding images to style elements 388
adding logo to output 454
inserting images with style templates 482
modifying titles/footnotes 392

RTF control strings 484
PREPAGE= option, ODS RTF statement
138–139
PREPEND option, ODS PATH statement
486–488
PRETEXT= attribute 484
PRINT media type 472–473
PRINT procedure
column heading style attributes 532–533
creating output data sets 232–233
custom style elements 479
HTML class attribute 476
LABEL statement 56–57
ODS OUTPUT CLOSE statement and 228
output examples 528–529
STYLE= option 122, 158, 476, 532, 542
SUM statement 536
SUMBY statement 536
table cell style attributes 534–535
table templates and 302
TITLE statement 336
totals style attributes 536–537
VAR statement 476, 534
Printer Control Language
See PCL
printer drivers 178–179, 410
PRINTER= option, ODS PRINTER statement 174, 178, 192
Printer style template
as default 346
identifying font definitions 400–401
modifying default font sizes 402–405
overview 394–397
removing table borders/rules 424–425
simulating listing output 434
spacing inside tables 426
PRINTERPATH= system option 172
PrinterSmaller style template 404–405
PRINTTO procedure 208
PROC DOCUMENT statement 312–313
PROC FORMAT statement 208
PROC PRINT statement 122, 158, 476, 532, 542
PROC REPORT statement
BOX option 512
COLWIDTH= option 508–509
OUT= option 232, 506
SPANROWS option 512
STYLE= option 122, 158, 494, 510, 542
PROC TABULATE statement
OUT= option 528
STYLE= option 122, 158, 519, 542
traffic lighting technique 524–525
procedure labels
displaying in bookmarks 154
modifying in table of contents 42–43
modifying multiple 44–45
procedure titles 436–437
procedures
See also multiple procedures

building output objects 200
combining results 30–31, 226–227
comparing output data sets to results
214–215
creating output data sets 218–219, 232–233
graphics output and 546
labels in table of contents 42–43
modifying fonts for 416–417
NOPRINT option 224
ODS output and 208–209
ODS OUTPUT CLOSE statement and 228
style definitions 16
table definitions 14, 16, 302
Program Data Vector (PDV) 256, 260, 262
PROTECTSPECIALCHARS style attribute 124–127, 280, 484
PS
See PostScript files
PS option, ODS PRINTER statement 176, 179, 410
PS printer style 394
PUT_PAGE technique 270
PUT statement
creating ODS output 256–257
modifying ODS output 266–267
ODS option 256, 260, 272
tagset template event code 333
%PUT statement 491
PUTN function 104
Pyx destination 322

Q

QUIT statement 228
quotation marks 494
%QUOTE macro function 62

R

RAW method, ODS ESCAPECHAR statement 134
RBREAK statement, REPORT procedure 500
READ option, ODS PATH statement 486
REG procedure
creating graphic images 548
graphics output example 546–547
output objects and 200, 202
regression plots 550
Rehabilitation Act 453
REMOVE option, ODS PATH statement 488
RENAME= option, ODS OUTPUT statement 230
REPARCHIVE tagset 332–333
REPLACE statement, TEMPLATE procedure
modifying font size 403
modifying table borders 420
modifying typefaces 408
overview 362
REPORT procedure
See also COMPUTE statement, REPORT procedure
See also DEFINE statement, REPORT procedure
adding graphics to tables 504–505
adding traffic lighting 502–503
BOX option 512

BREAK statement 500, 510–511
column heading style attributes 494–495
COLWIDTH= option 508–509
controlling column widths 508–509
creating cleaner output 512–513
creating output data sets 232–233, 506–507
custom style elements 479
embedding graphs 570–571
HTML class attribute 476
inserting blank lines 510–511
ODS LISTING statement and 506
OUT= option 232, 506
RBREAK statement 500
row style attributes 498–499
SPANROWS option 512
STYLE= option 122, 158, 494, 510, 542
subtotals style attributes 500–501
table cell style attributes 496–497
table templates and 302
totals style attributes 500–501
reports
See also specific types of output
custom 271–291
multi-page 100–101
RESET option, ODS PATH statement 488
Results window
docking 296
output example 228–229, 312–313
replaying document stores 300–301
tracking output objects 18–19
turning off results delivery 226–227
RIGHTMARGIN= style attribute 378
RIGHTMARGIN= system option 96, 186, 432
Risk Dimension 546
row heading style attributes 514–515
row style attributes 498–499, 520–521
RowFooterEmphasis style element 400
RowHeader style element 370–373, 384–387, 400
RowHeaderEmphasis style element 400
RSREG procedure 230
RTF destination output
adding customized dates 104–105
adding dynamic dates 102–103
adding hyperlinks 114–115, 148–149
adding images 108–109, 460–461, 464–465
adding images to each page 462–463
adding images to style elements 388–389
adding logo to output 454
adding more complex text 112–113
adding static dates 102–103
adding text 110–111
aligning decimals 122–123
applying CSS file 468–469
as output destination 14
comparing listing output 434–435
control strings 104–105, 120–121, 484–485
control words 128–129, 134–135

controlling column widths 530
controlling page breaks 94–95
controlling page margins 96–97, 432–433
controlling page numbering 100–101, 130–131
controlling page orientation 90–91
controlling page size 92–93
creating files 82–83
creating multiple columns 98–99
DATA step processing 256–257
differences in style 396–397
enhancing with field codes 128–129
examples 9–10, 18–19
external image files 558
graphics output and 548, 558–561, 568
handling page breaks 88–89
identifying available fonts 410–411
importing files into word processors 84–85
importing into slide show 132–133
indenting values 540–541
inserting field codes 126–131
inserting images with style templates 482
inserting text with PREPAGE= option 138–139
justification style attribute 122, 268
limiting tags in 58–59
line breaks 104–105
Microsoft Word considerations 124–125
modifying font sizes 406–407
ODS CLOSE statement and 228
overview 81
PDF output comparison 396
placing title/footnote text 106–107
reducing output file size 426–427
replaying document stores 300–301
spacing inside tables 426
specifying metadata 120–121
style definitions 357
table headers 144–147
table of contents 42, 116–119, 142–143
TAGSETS.RTF destination 114, 136–143
working with headers and footers 86–87
RTF style template
as default 346
identifying font definitions 400
modifying font definitions 412–413
modifying font sizes 406–407
overview 396–397
RTFImage style 460, 462
RTFSmaller style template 406–407
RULES= style attribute 380, 382, 422–425
run-group processing 298
RUN statement 228

S

SAS Enterprise Guide 486
SAS/ETS 546
SAS Explorer 214–215, 296
SAS/GRAPH

See also graphics output
additional resources 572–573
batch mode tips 298
changing output via options 556–557
creating graphics output 543
creating HTML output 554–555
creating RTF output 94
document store considerations 303
external image files 558
graphics procedures 28
running legacy code 560
sizing options 559
Zdevices and 545
SAS/HPF 546
SAS item stores 273, 294
SAS logs
CSV tagset information in 324
output example 491–492
referencing table templates 271
SAS option, ODS PRINTER statement 178, 410
SAS/QC 546
SAS Registry, viewing 92
SAS/STAT
batch mode tips 298
external image files 558
graphics output and 546
SASDATE option, ODS RTF statement 102
SASEDOC LIBNAME engine 318–319
SASEMF device driver 558
SASGDGIF module 560
SASGDIMG module 560
SASGDTIF module 560
SASHELP.TMPLMST item store
consistency in using 303
overview 322
parent style template 467
removing access to templates 488
simulating listing output 434
storing style templates 354, 357
storing table templates 273
TAGSETS folder 322
SASHELP.VSTYLE view 490
SASPRTC device driver 558
SASUSER.TEMPLAT item store 338, 354, 488
SASWEB.CSS file 474–475
SASWEB style
HTML output 15, 50–51
LISTING destination 510
MARKUP destinations 326
REPORT procedure 500
SCREEN media section (CSS) 472–473
<SCRIPT> tag 62
scripts
adding to output 62
startup 486
SCSCATTER procedure 550–551
SDATE macro variable 104
SELECT action

ODS HTML statement 244, 250
ODS RTF statement 250
SELECT option, ODS PRINTER statement 189
SET statement 260, 268
SETLABEL statement, DOCUMENT procedure 312–313
SGML (Standard Generalized Markup Language) 321
SGPANEL procedure 544, 550–551
SGPLOT procedure 544, 550–551
SGRENDER procedure 544, 552–553
SGSCATTER procedure 544
SHORT_MAP tagset 330
SHOW action, ODS RTF statement 248
Show/Hide paragraph marker 118
SHOW option, ODS PATH statement 487–488
ShutDownFunction style element 374–375
SKIP option, BREAK statement (REPORT) 510–511
slash (/) 321, 496
slide shows, importing RTF output into
132–133
SORT procedure 208
SOURCE statement, TEMPLATE procedure
modifying font sizes 404–407
modifying table borders 420
numbers in table of contents 428
viewing source code 350–351, 358–361
spanning column headers 278–279
SPANROWS option, PROC REPORT statement 512
special characters
in markup process 321
turning off protection 126, 134, 280, 484
spreadsheets
Excel Spreadsheet Markup Language 14, 18, 78–79
importing ODS output 78–79
tagset support 328
SQL procedure 490
square brackets [] 494, 514, 532
SRC= attribute 456
Standard Generalized Markup Language (SGML) 321
StarOffice, importing files 84–85
START= attribute 278
STARTPAGE= option
ODS ESCAPECHAR statement 110
ODS PDF statement 166, 170
ODS PRINTER statement 464–465
ODS RTF statement 94, 242
startup scripts 486
StartupFunction style element 374–375
Statdoc style template 348
STATGRAPH graph templates 544
static dates, adding 102–105
statistical graphics
SGPANEL procedure 544, 550–551
SGPLOT procedure 544, 550–551
SGRENDER procedure 544, 552–553
SGSCATTER procedure 544
storage considerations
graph templates 357
output object information 303

style templates 354, 357
table templates 273, 357
tagset templates 273, 357
updated templates 486–487
STORE= option, DEFINE TAGSET statement (TEMPLATE) 333
StrongFont attribute 401
style attributes
See also specific style attributes
changing on the fly 493
changing with PRINT procedure
532–537, 542
changing with REPORT procedure
494–501
changing with TABULATE procedure 514–515, 518–523, 542
defined 350
indenting values 538–539
modifying 378–379
naming groups 357
traffic lighting technique 282
viewing 358
style class 350
style definitions
See style template definitions
STYLE_DISPLAY tagset 330
style elements
See also specific style elements
adding images 388–389
color control and 440–441
defined 350, 357
documentation for 371, 476
inheritance and 358–361, 370–371, 380, 384
modifying footnotes with 390–393
modifying specific 372–373
modifying titles with 390–393
STYLE function, ODS ESCAPECHAR statement 60–61, 160
STYLE= option
See also specific style templates
ALL option, TABLE statement (TABULATE) 522
BOX= option, TABLE statement (TABULATE) 516, 526
BREAK statement, REPORT procedure 500
CLASS statement, TABULATE procedure 514, 520
CLASSLEV statement, TABULATE procedure 520, 538
COMPUTE statement, REPORT procedure 504
DEFINE statement, REPORT procedure 122, 158, 496–499
DEFINE statement, TEMPLATE procedure 350–353
finding style definitions 354
listing style template definitions 346
modifying style definitions 352

modifying table borders 420–421
ODS HTML statement 490
ODS PDF statement 348
ODS RTF statement 82–83, 432
PRETEXT= attribute 122
PROC PRINT statement 122, 158, 476, 532, 542
PROC REPORT statement 122, 158, 494, 510, 542
PROC TABULATE statement 122, 158, 542
SUM statement, PRINT procedure 536
TABLE statement, TABULATE procedure 524
VALUE statement, FORMAT procedure 540–541
VAR statement, PRINT procedure 534
VAR statement, TABULATE procedure 514, 520, 526
STYLE_POPUP tagset 330, 358, 364
STYLE statement, TEMPLATE procedure 280, 370–373, 402–403
style statements 350
style template definitions
See also specific style templates
abstract style elements 370–371
adding images to style elements 388–389
adding logo to output 454–455
altering HTML frames 376–377
built-in 345–357
changing output via 556–557
color-coordinated 452–453
color schemes 448
color style elements 364–365
consistent column widths 182
controlling page margins 96
creating custom 434–435
CSS page breaks 38
data cell-related style elements 384–387
default style 358–363
defined 357
differences in RTF style 396–397
document stores and 303
examining components 350–351
finding custom 354–355
font style elements 366–367
graphics output and 544–545, 554, 558
header-related style elements 384–387
HTML Body style attributes 378–379
HTML class attribute 476
HTML style element 368–369
in item stores 294

inserting images 482–483
making smaller output files 480–481
matching corporate style guidelines
470–471
modifying 348–349, 352–353
modifying footnotes 390–393
modifying specific style elements
372–373
modifying titles 390–393
overview 16–17
PREIMAGE= attribute 108
printer output and 180, 394–395
removing access 488
replacing fonts in 408–409
RTF control strings 484–485
shutdown style elements 374–375
specifying alternates 490–492
style template definitions (*continued*)
startup style elements 374–375
storing 354, 357
STYLES.DEFAULT 28
switching between 348–349
syntax changes 345
table style elements 380–383
text style element 368–369
turning off special character protection 126, 134, 280
viewing available 346–347
STYLESHEET= option
ODS HTML statement 474
ODS TAGSETS.ODSSTYLE statement 364
SUBJECT= option, EMAIL engine 76
subtotals style attributes 500–501, 522–523
SUFFIX1= style attribute 368
SUM statement, PRINT procedure 536
SUMBY statement, PRINT procedure 536
SUMMARY procedure 528
SUPER function, ODS ESCAPECHAR statement 60–61
SUPERFMT format 60–61
superscripts, adding 60–61
SXLE LIBNAME engine 321
SYMBOLGEN system option 491
SYSDATE automatic macro variable 287
%SYSFUNC macro function 104
SYSPRINT= system option 172
SYSSCPL automatic macro variable 287
system options
See also specific system options
affecting ODS PRINTER output
190–191

graphics output and 544
replaying document stores and 303
SYSTEM_TITLE event 330
SystemFooter style element
adding colors to color scheme 450
adding images to output 462
creating font definitions 415
modifying 392
SystemTitle style element
font attributes 366–367
inheritance and 358–361, 364–365
modifying 392
SystemTitle2 style element 392

T

table attributes 273, 288–291
table borders
changing spacing inside 426–427
manipulating 16
modifying 420–421
removing 422–425, 512
style definitions 14
table cells
changing style attributes 496–497, 518–519, 534–535
changing style elements 384–387
table colors, modifying 448–449
table definitions
See table template definitions
table headers
modifying 144–145
rotating/aligning 146–147
TABLE_JUST template 288
table of contents
adding to RTF output 116–117
appending to existing 72–73
completing 118–119
drill-downs in output tables 66–67
frame file and 34–35
generating 32–33
hyperlinks and anchors 52–53
modifying procedure labels 42–43
modifying style elements 376
removing numbers from 428–429
setting fixed width 430–431
table of pages 36, 52–53
table row style attributes 498–499
table rules, removing 422–425
TABLE statement, TABULATE procedure
ALL option 522
BOX= option 516, 526
controlling column widths 530
STYLE= option 524
Table style element
changing 380–384
differences in printer style 394
removing table borders/rules 424–425

setting width for HTML output 430
Table table template 272
<TABLE> tag 26–27, 68–69
table template definitions
calculating columns 276–277
consistent column widths 182
creating custom templates 274–275,
280–281, 478–479
document stores and 303
dynamic variables in 288–291
GENERIC column attribute 272–273
generic columns in 284–285, 288
in item stores 294
JUSTIFY attribute 273, 288–291
macro variables in 286–287
ORDER_DATA attribute 273
overview 16–17, 271–273
procedures and 14, 16, 302
removing access 488
spanning column headers 278–279
storing 273, 357
traffic lighting technique 282–283
table values, adding traffic lighting 502–503
TABULATE procedure
adding graphics to tables 526–527
CLASS statement 514, 520, 530
CLASSLEV statement 520, 538
column heading style attributes 514–515
controlling column widths 530–531
creating output data sets 506, 528–529
custom style elements 479
handling page breaks 88–89
HTML class attribute 476
indenting values 540–541
OUT= option 528
row heading style attributes 514–515
row style attributes 520–521
STYLE= option 122, 158, 519, 542
subtotals style attributes 522–523
table cell style attributes 518–519
TABLE statement 516, 522–527, 530
table templates and 302
totals style attributes 522–523
traffic lighting technique 524–525
VAR statement 514, 520, 526, 530
TAGATTR= style attribute 280, 374
TAGSET= option, ODS MARKUP statement 322
tagset templates
applying custom 336–337
checking for updates 338–339
creating 321–323
creating CSV files 324–325
creating custom tagsets 321, 334–335
creating HTML result files 326–327
creating XML files 328–329
diagnostic tagsets 330–331, 358
downloading 338
modifying events 332–333

overview 16, 321
removing access 488
storing 273, 357
TAGSETS folder 322
TAGSETS.RTF destination
adding hyperlinks to RTF output 114, 148–149
graphics output and 558
modifying table headers 144–145
overview 136–137
PREPAGE= option 138
<TD> tag 26–27, 68–69
template definitions 333
See also style template definitions
See also table template definitions
TEMPLATE procedure
See also CLASS statement, TEMPLATE procedure
See also DEFINE statement, TEMPLATE procedure
See also PARENT= statement, TEMPLATE procedure
See also SOURCE statement, TEMPLATE procedure
adding images to style elements 388
altering HTML frames 376
CELLSTYLE-AS statement 282
COLUMN statement 273, 278
creating custom tagsets 334–335
creating font definitions 414–415
creating style templates 382–387
creating tagset templates 321–323
CSS page breaks 38
DEFINE EVENT statement 333
DEFINE HEADER statement 272, 278
DEFINE STYLE statement 430
DEFINE TAGSET statement 333
defining styles 372–374
DYNAMIC statement 289
graphics output support 546
HTML Body style attributes 378
LIST statement 354
LIST STYLES statement 346, 354
modifying font definitions 412–413
modifying output fonts 56–57
modifying titles and footnotes 390–393
MVAR statement 286–287, 332
NMVAR statement 286
removing lines 42
removing table borders/rules 424–425
REPLACE statement 362, 403, 408, 420
TEMPLATE procedure (*continued*)
STYLE= statement 280, 370–373, 402–403
VARIABLES= option 290
TEMPLATE= suboption, FILE PRINT ODS statement 274, 290
text
adding annotations 160
adding complex 112–113
adding to RTF output 110–111
assigning to dynamic variables 284
combining graphs with 570–571
inserting with PREPAGE= option 138–139

modifying hyperlink color 114
text editors
See word processors
TEXT_MAP tagset 330
TEXT= option, ODS RTF statement 110, 114, 130
text style element 368–369
THEN statement 282, 502
TIMEPLOT procedure 416
title bars, changing 46–47
TITLE= option
ODS HTML statement 27, 46–47, 80
ODS RTF statement 120, 124
TITLE statement
adding control words 128–129
adding dynamic dates 104
adding escape sequence 100
adding field codes 128–130
adding images to output 462, 465
BOLD option 56
COLOR= option 56
FONT= option 56
HEIGHT= option 56
hiding procedure titles 436
HTML tags in 56–57
inserting field codes 127
inserting images with style templates 482
ITALIC option 56
JUST= option 56
JUSTIFY=RIGHT option 104
LINK= option 40, 56
modifying title bar 46–47
page numbering 130–131
script support 62
style definitions in 350–351
XML output and 336
TITLE1 statement 392
TITLE2 statement 42, 104
<TITLE> tag 26–27
TitleAndNoteContainer element 396
TitleFont attribute 366–367, 400–401
TitleFont2 attribute 401
titles
See also TITLE statement
adding annotations 160
after copying output objects 316–317
default area for 108
for DOCUMENT destination 310–311
inserting logos 108
modifying 116
modifying color of 442–443
modifying default font sizes 402–403
modifying using style elements 390–393
placing text in document 106–107
procedure 436–437
style definitions in 350–351
TITLES option, FILE PRINT ODS statement 270
TitlesAndFooters style element
adding background images 458

controlling color 440
inheritance and 358, 364–365
modifying titles and footnotes 390–394, 442–443
TO= option, EMAIL engine 76
TOC_DATA option, ODS RTF statement 116, 118
TOPMARGIN= system option 96, 186, 432
totals style attributes 500–501, 522–523, 536–537
TPL extension 338
TPL_STYLE_LIST tagset 330
TPL_STYLE_MAP tagset 330
<TR> tag 26–27, 68–69
traffic lighting technique 282–283, 502–503, 524–525
TRHDR suboption, ODS TAGSETS.RTF statement 144, 146
TROWD suboption, ODS TAGSETS.RTF statement 144, 146
TROWHDRCELL suboption, ODS TAGSETS.RTF statement 144, 146
TTEST procedure 226, 228
TYPE= option, ODS MARKUP statement 322
typeface, font 408–409, 494

U

UNDERLINE= table attribute 289
UNIFORM option
ODS PDF statement 182
ODS PRINTER statement 182
UNIVARIATE procedure 546–548
UNIX operating system 577
UPDATE option, ODS PATH statement 486
updates, tagset templates 338–339
URL= option
ODS ESCAPECHAR statement 114, 158
ODS HTML statement 28, 74, 80, 554
user-defined formats 302, 503

V

VAL keyword 282
VALUE statement, FORMAT procedure 540–541
VAR statement, PRINT procedure 476, 534
VAR statement, TABULATE procedure
controlling column widths 530
STYLE= option 514, 520, 526
variables
See also macro variables
dynamic 284, 288–291
indenting values 538–539
numeric 272, 284–285
selecting for output 260
VARIABLES= option
FILE PRINT ODS statement 260–262, 264
TEMPLATE procedure 290
VBAR statement, GCHART procedure 28
VBOX statement, SGPLOT procedure 550
Viewtable (SAS Explorer) 214–215
VSIZE option, GOPTIONS statement 559

W

W3C consortium 474, 476

WATERMARK= style attribute 388
Web browsers
color considerations 444–445
CSS page break command 38
graphics output 28
hot spots 562–563
modifying title bar 46–47
Web pages
creating output with just results 68–69
creating with table of contents 72–73
drill-downs in output tables 66–67
hit counters 62
HTML files 34–35
hyperlinks and anchors 52–53
Web sites
hyperlinks and anchors 54–55
moving ODS output into 74–75
WHERE conditions
CELLSTYLE-AS statement 282
COMPUTE-AS statement and 276
restricting queries 490
selecting 250–253
WHERE= option
LIST statement, TEMPLATE procedure 354
ODS HTML SELECT statement 250
ODS RTF SELECT statement 250
whisker plots 550
WiderBorder style template 420
WIDTH= option
DEFINE statement, REPORT procedure 508–509
DEFINE statement, TEMPLATE procedure 430
WIDTH= style attribute 508–509, 530
Windows operating system 576
Windows Photo Gallery 546–547
WML tagset 328
WMLOLIST tagset 328
word processors
completing table of contents 118–119
difficulties importing listing output 6–7
importing files 84–85
RTF output 124–125, 136–137
WORDDATE format 104
WordPerfect, importing files 84–85
WORDSTYLE option, ODS RTF statement 138
WORK library 273, 302, 486–487
WORK.ORIGDOC document store 294, 298, 308
WORK.TEMPLAT item store 434
WRITE option, ODS DOCUMENT statement 294

X

XAXISOPTS option 552
XBRL tagset 328
XMAX option, GOPTIONS statement 559
XML (Extensible Markup Language) 321
XML output
creating with ODS MARKUP statement 328–329
custom tagset templates 336–337
importing into spreadsheets 78

output example 14
special characters 321
XML tagsets 328, 334–335
XPIXELS option, GOPTIONS statement 559, 566

Y

YAXISOPTS option 552
YMAX option, GOPTIONS statement 559
YPIXELS option, GOPTIONS statement 559, 566

Z

z/OS operating system 578–580
Zdevices 545
ZGIF device driver 545, 560
ZJPEG device driver 560
ZPNG device driver 560

Symbols

& (ampersand) 333
  (HTML) 538
<> (angle brackets) 280, 321–323
@media section (CSS) 472–473
\ (backslash) 126, 134, 484
^ (caret) 148, 540
: (colon) 324
, (comma) 324
{} (curly braces)
enclosing style attributes 494, 514, 532
in commands 321
special character protection and 126
() (parentheses) 364
. (period) 321
(pound sign) 274, 318
/ (slash) 321, 496
[] (square brackets) 494, 514, 532

---

[1] This file has been simplified for demonstration purposes. A number of tags that control the procedure title have been removed.

[1] Earlier versions of SAS, such as SAS 8.2, were designed to work best with Word 97 and later.

[1]Terminology can get confusing here. Table templates and table definitions are the same thing. When the ODS TRACE statement identifies a “template” in its output, it is referring to the table definition or table template that controls how the output is displayed.

[3] There are other font weights you can use, such as Light and Extra-Bold, but these might not be supported by all fonts. A list of font weights, as well as font styles and font widths, is included in the SAS documentation.

[1] To learn which style attributes to use, go to the *SAS Output Delivery System: User’s Guide* and find the section on PROC TEMPLATE. Then go to the section on the DEFINE STYLE statement. It has a lengthy section on style attributes that lists each of the attributes and what parts of the output they affect.