

PROC DOCUMENT by Example Using SAS®

Michael Tuchman

SAS® Press





PROC DOCUMENT by Example Using SAS®

Michael Tuchman

SAS® Press



PROC DOCUMENT by Example Using SAS®

Michael Tuchman



support.sas.com/bookstore

The correct bibliographic citation for this manual is as follows: Tuchman, Michael. 2012. *PROC DOCUMENT by Example Using SAS®*. Cary, NC: SAS Institute Inc.

PROC DOCUMENT by Example Using SAS®

Copyright © 2012, SAS Institute Inc., Cary, NC, USA

ISBN 978-1-61290-096-4 (electronic book)

ISBN 978-1-60764-667-9

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.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

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 27513-2414

1st printing, October 2012

SAS Institute Inc. 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 Books Web site at support.sas.com/bookstore 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.

614659rlh29Jan2013

To my family and to Scott Hecht,
whose unflagging support made
this book possible.

Contents

[About This Book](#)

[About The Author](#)

[Acknowledgments](#)

[Chapter 1: Introduction and Quick-Start Guide](#)

[What Is an ODS Document?](#)

[Life Before ODS Document](#)

[Life with ODS Document](#)

[Why Use an ODS Document?](#)

[Why Not Manage Change with Office Software?](#)

[QuickStart Guide](#)

[File Location](#)

[Data](#)

[Saving Output to an ODS Document](#)

[Replaying Output.](#)

[Subsetting Output](#)

[Managing Output](#)

[Changing Titles and Footnotes](#)

[Using Folders](#)

[Deleting and Overwriting Document Contents](#)

[What Remains to Be Covered?](#)

[Summary](#)

[Chapter 2: The ODS DOCUMENT Destination](#)

[Introduction](#)

[Convention](#)

[What Is the Output Delivery System \(ODS\)?](#)

[Definition of the ODS Destination](#)

[ODS without the Document Facility](#)

[ODS with the DOCUMENT Destination](#)

[Role of the DOCUMENT Procedure](#)

[Reasons to Use the Document](#)

[Naming ODS Output](#)

[The ODS TRACE Statement](#)

[Determining the Directory Where ODS Output Is Stored](#)

[Implementation Details of the Item Store](#)

[Controlling Output Using SELECT and EXCLUDE Lists](#)

[Summary](#)

[Chapter 3: The ODS DOCUMENT Statement](#)

[Introduction](#)

[The ODS DOCUMENT Statement](#)

[The NAME= Option](#)

[Example: Storing Output at the Top Level](#)

[Comparing ODS TRACE Output to ODS Document Paths](#)

[The DIR= Option](#)

[Behavior of the Pathname](#)

[The CATALOG= Option](#)

[Using Access Options \(Write and Update\)](#)

[Example: Reviewing Your Progress](#)

[Summary](#)

[Chapter 4: Listing Documents Using the DOCUMENT Procedure](#)

[Introduction](#)

[The DOCUMENT Procedure](#)

[Terminology](#)

[Directory Notation](#)

[Root Level Directory](#)

[Current Directory](#)

[Invoking the DOCUMENT Procedure](#)

[The NAME= Option](#)

[The LABEL= Option](#)

[Running the Document Procedure Interactively](#)

[Access Modes](#)

[The LIST Statement](#)

[Simple Invocation of the LIST Statement](#)

[Sequence Numbers](#)

[LEVELS= Option](#)

[Using the DETAILS= Option](#)

[ORDER= Option](#)

[BYGROUPS Option](#)

[Warning: BYGROUPS Option Might Show No Output](#)

[Using WHERE= Clauses to Conditionally View a Document](#)

[WHERE Variables](#)

[Examples](#)

[Interlude](#)

[Debugging WHERE= Clauses](#)

[The WHERE= Clause and ODS Names](#)

[Comparison with the ODS SELECT and EXCLUDE Statements](#)

[LABELPATH_ - Proceed at Your Own Risk DETAILS option is not](#)

The SETLABEL Statement

Important: Labels Are Not Titles

Summary

Chapter 5: The REPLAY Statement

Introduction

How Replay Works

REPLAY Statement Syntax

Replaying to All Open Destinations

The DEST= Option

Replaying to Multiple Destinations

Replaying in a Different Order

The LEVELS= Option

Overriding Titles with the ACTIVETITLE and ACTIVEFOOTN Options

Replaying Using WHERE= Clauses

Replaying Subsets of Output Objects

Example: Reordering BY-Grouped Output

Replaying Output with Temporary Formats

Temporary or Permanent Formats?

Summary

Chapter 6: Managing Folders

Introduction

How ODS Document Folders are Different

Absolute Pathnames

Relative Pathnames

Positioning Arguments

When the TO Argument Requires a Sequence Number

WHERE= Clauses and Directories

The MAKE Statement

The DIR Statement

The RENAME TO Statement

Renaming a Document

Warning about Renaming Output Objects

The COPY TO Statement

A Common Mistake

Copying Using the LEVELS= option

The MOVE TO Statement

Browsing Collections of Documents with the DOC Statement

The LIBRARY= Option

The NAME= Option

[The DELETE Statement](#)

[Warning: You Cannot Delete the Current Directory](#)

[The HIDE Statement](#)

[Extended Examples](#)

[Replay with the LEVELS= Option](#)

[Listing Every Document in Your System](#)

[Description of %alldocs Macro Arguments](#)

[Managing ODS Documents via the Graphical User Interface.](#)

[Summary](#)

[Chapter 7: Customizing Output](#)

[Introduction](#)

[The Document Used in This Chapter](#)

[Output Objects in an ODS Document](#)

[Introduction to Context](#)

[Example: A Context Reference Chart](#)

[The OBPAGE Statement](#)

[The OBPAGE Statement Might Be Ignored in Some Procedures.](#)

[The OBTITLE Statement](#)

[The SHOW option.](#)

[The OBSTITLE Statement](#)

[Using #BYVAL and #BYVAR Variable Values to Build a Custom Subtitle](#)

[Example: Using the #BYLINE Variable Value](#)

[Object Notes: The OBANOTE and OBBNOTE Statements](#)

[Document Notes](#)

[Adding Notes Using the TEXT Statement](#)

[Adding Notes Using the DOCUMENT Procedure's NOTE Statement](#)

[Document Notes Compared with Object Notes](#)

[The Final Version of the Example Document](#)

[Summary](#)

[Chapter 8: Exporting to Data Sets](#)

[Introduction](#)

[The Power of Combining the ODS OUTPUT Destination with the Document Facility](#)

[The ODS OUTPUT Destination](#)

[The ODS OUTPUT Statement](#)

[The ODS OUTPUT Destination without the Document](#)

[The ODS OUTPUT Destination with the Document](#)

[Combining Data Sets from the Document](#)

Combining the OUTPUT and DOCUMENT Destination

ODS OUTPUT and the DOCUMENT Procedure's Data Sets

[Simple List Output](#)

[Contents of the Document Data Listing](#)

[Conditionally Executing the DOCUMENT Procedure with CALL EXECUTE](#)

The SASEDOC Engine

[Accessing a Directory of Output](#)

[Applications of the SASEDOC Engine](#)

[The SASEDOC Engine and Sequence Numbers.](#)

Summary

Chapter 9: Importing Data to the Document

Overview

[The Document Is Not a SAS Library](#)

[The IMPORT Statement](#)

[Data for Examples](#)

[Importing a Data Set](#)

[Importing Text Files](#)

[Importing Data and Its Template to a Document](#)

Summary

Chapter 10: Working with Links

[Introduction](#)

[Creating a Link](#)

[Source and Target](#)

[Replaying Links](#)

[The FOLLOW Option of the LIST Statement](#)

[Orphaned Links](#)

[Document Management Using Links](#)

[Modifying PDF Bookmarks Using Links](#)

[Hard Links](#)

Summary

Chapter 11: Working with Templates

[Introduction](#)

[What Is a Template?](#)

[Terminology](#)

[Template Store](#)

[Template Search Path](#)

[Template Access Mode](#)

[Updated Document for Examples in This Chapter](#)

[Defining Your Own Template Store](#)

[Template Store Organization](#)

[Exploring the Template System](#)

[Example: Removing Automatic Page Breaks](#)

[Modifying Template Definitions](#)

[Undoing Template Changes](#)

[Templates and the Document](#)

[Identifying Templates in the Document Listing](#)

[The OBTEMPL Statement](#)

[Document Saves a Reference, Not the Actual Template](#)

[Modifying the DOCUMENT Procedure's Templates](#)

[Styles](#)

[Assigning a Custom Template to an Object in a Document](#)

[The Custom Template](#)

[Data Set Used](#)

[Summary](#)

[Appendix](#)

[References](#)

[Index](#)

[Accelerate Your SAS Knowledge with SAS Books](#)

About This Book

Purpose

The purpose of this book is to make the SAS practitioner aware of functionality that makes managing SAS output easier. The DOCUMENT procedure has been part of the Output Delivery System for quite some time, and although not at all difficult, it needs to be more widely publicized. PROC DOCUMENT is a basic tool that SAS programmers of all levels should take advantage of. This book is intended to show you how to get started quickly while learning the more advanced features as you need them.

Is This Book for You?

This book is for you if you create SAS output and want to have the opportunity to re-present it or re-combine it with other SAS output, all without re-running the original analytical code that created the output.

Prerequisites

This book assumes that you access the SAS System through code and feel comfortable with DATA step programming. It also assumes that you have basic experience with the Output Delivery System (ODS), at least how to send SAS procedure output to an ODS destination. The book defines the technical aspects of ODS needed to use the DOCUMENT procedure effectively. The material in Chapter 4 regarding WHERE= options makes analogies with the WHERE= data set option. It would be helpful to be comfortable with that as well. This book references a couple of SAS/STAT procedures, but only for the purpose of demonstrating a variety of SAS procedure output. Understanding what they do is not critical for understanding the discussion of PROC DOCUMENT.

Scope of This Book

All DOCUMENT procedure statements in SAS 9.3 are demonstrated, as well as the necessary terminology and concepts from ODS needed to understand and apply the examples.

Output and Graphics Used in This Book

The output used in this book is output from the PDF and HTML destinations. These are part of standard ODS. Graphical output is produced by ODS Statistical Graphics, including the SGPlot and SGPANEL procedures.

Author Page

You can access this book's author page at support.sas.com/tuchman. For features that relate to this specific book, look for the cover image of this book. The links below the cover image will take you to a free chapter, example code and data, reviews, updates, and more.

Example Code and Data

This book uses two ODS documents as sources for its examples. You can build these documents using only the code in this book and data sets that are shipped with SAS as part of the SASHELP library. The ODS DOCUMENT statements used to build the documents for the examples are also stored in a separate section. This should make it easier to rebuild either of the documents to the state they should be in to start a particular chapter.

You can access the example code and data included with this book by linking to its author page at support.sas.com/tuchman.

For an alphabetical listing of all books for which example code and data 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.

Additional Resources

SAS offers a rich 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, we have learning products that can help you maximize your investment in SAS.

| | |
|----------------------------|--|
| Bookstore | support.sas.com/publishing/ |
| Training | support.sas.com/training/ |
| Certification | support.sas.com/certify/ |
| Higher Education Resources | support.sas.com/learn/ |
| SAS OnDemand for Academics | support.sas.com/ondemand/ |
| Knowledge Base | support.sas.com/resources/ |
| Support | support.sas.com/techsup/ |
| Learning Center | support.sas.com/learn/ |
| Community | support.sas.com/community/ |
| SAS Forums | communities.sas.com/index.jspa |
| User Community Wiki | www.sascommunity.org/wiki/Main_Page |

Comments or Questions

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

Mail: SAS Institute Inc.

SAS Press

Attn: Michael Tuchman

SAS Campus Drive

Cary, NC 27513

Email: saspress@sas.com

Fax: (919) 677-4444

Please include the title of this book in your correspondence.

SAS Book Report

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

About The Author



Michael Tuchman is a senior statistician at Accolade, Inc., a leading provider of Professional Health Assistant services. A SAS user since 1999, Tuchman applies his extensive knowledge of SAS to understanding how customers utilize healthcare, ultimately empowering them to get more value out of the healthcare system. Prior to joining Accolade, he worked at SDI Health (now part of IMS Health) where he mined healthcare data to find patient behavior patterns. Prior to joining SDI, he developed predictive models in the Health Informatics Department at Aetna. Tuchman received a BS from the Massachusetts Institute of Technology, and an MS in mathematics from the University of North Carolina at Chapel Hill.

Learn more about this author by visiting his author page at support.sas.com/tuchman. There you can download free chapters, access example code and data, read the latest reviews, get updates, and more.

Acknowledgments

I am grateful to everybody who had a hand in bringing this book to fruition. Starting with the text itself, there were numerous people who provided insightful feedback on several drafts of this material. David Kelley and Bari Lawhorn, on the development team of PROC DOCUMENT, were a constant source of valuable information about which features of the Document Facilities I should emphasize. I am equally grateful to all of the SAS technical reviewers whose comments made this book better, which includes Mary Harding, Mike Kalt, Kathryn McLawhorn, Chevell Parker, Kevin Smith, and Cynthia Zender. In addition to providing literary support, David and Bari provided technical support about PROC DOCUMENT.

Thanks to those who provided additional feedback and support during the writing of the first draft of the book. Vicki Boykis provided feedback, free of charge, about several of the early chapters in the book. Although the organization of the material has changed beyond recognition since those drafts were reviewed, her suggestions on the narrative flow of Chapter 1 remain evident.

There is one person who deserves special mention, and that is Scott Hecht. Scott served as an external reviewer for the book. In addition to acting in this capacity, he reviewed every single chapter before I submitted it to SAS Press. If that were all he did, I would be extremely grateful, but he also was the person I went to when I wanted to explore new material or ponder the organizational and logical challenges that come with writing a technical book. I am particularly grateful to him for suggesting that we include the quick-start guide in the first chapter. Scott served as muse, editor, and friend. Every author should be as lucky as I was to have someone like him on his side.

For the appearance of the book, I would like to thank Caroline Brickley for her incredibly insightful job of copyediting and making numerous helpful suggestions to the text that I also look forward to applying in future works. I would also like to thank Candy Farrell for formatting the book and making all of the text corrections. I am grateful to the graphic designers, Jennifer Dilley and Marchelina Waugh for their roles in providing visual organization to the text and cover.

This book would be just a stack of paper if not for the marketing team's efforts in spreading the word. In particular, I am grateful to Stacey Hamilton and Aimee Rodriguez for coordinating the marketing of the book and to Shelly Goodin for her guidance on the proper use of social media, critically important for how technical communication is disseminated in today's world.

A new author has to learn about not only the optimal presentation of ideas, but also the process of how a manuscript is published. John West, my developmental editor, explained all of that to me and continued to encourage me as the manuscript made its way from draft form to final submission. He was also the epitome of patience despite the occasional slipped deadline. Thanks also go out to the SAS Press managing editor Mary Beth Steinbach for keeping the entire process moving smoothly. My acquisitions editor, Shelley Sessions, was encouraging and friendly while I submitted the book proposal.

In addition, of course, a tremendous hug to my family for being supportive of my many "basement weekends."

Chapter 1: Introduction and QuickStart Guide

[What Is an ODS Document?](#)

[Life Before ODS Document](#)

[Life with ODS Document](#)

[Why Use an ODS Document?](#)

[Why Not Manage Change with Office Software?](#)

[QuickStart Guide](#)

[File Location](#)

[Data](#)

[Saving Output to an ODS Document](#)

[Replaying Output.](#)

[Subsetting Output](#)

[Managing Output](#)

[Changing Titles and Footnotes](#)

[Using Folders](#)

[Deleting and Overwriting Document Contents](#)

[What Remains to Be Covered?](#)

[Summary](#)

What Is an ODS Document?

ODS stands for Output Delivery System, which is a system for producing SAS output in a variety of formats such as HTML, PDF, XML, LaTeX, as well as many others.

An ODS document, together with the DOCUMENT procedure, comprises a facility that enables you to store ODS output and reprint it at a later time. The goal of this chapter is to show you why this is such a powerful ability.

To see how easy it is to get started using the DOCUMENT procedure, review the examples in the quick-start guide in this chapter.

This ability to reproduce output without re-running SAS code is at the heart of the Document Facility. To see why this is useful, consider the following two scenarios.

Life Before ODS Document

Suppose your boss wants a summary of last week's results for the company's intranet site, and wants them displayed using the company's style standards. To reproduce this output, you try to find the analysis data set that was used to create the original results. You might need to search among many similarly named data sets in order to find the right one. Then, you need to find the version of the code that created the published results. You hope that this is the latest version of your code, but it might not be. Although you can re-run the process, you feel concerned about spending valuable time duplicating existing work.

Your boss arrives at your cubicle and asks whether you are ready. "Just a few more minutes," you timidly reply, wishing that you had been ready earlier.

Life with ODS Document

Now, imagine this scenario: When you ran your SAS program originally, you had the foresight to save all of your procedure and graph output to an ODS document. You did this knowing that you might need to access the same output in the future without re-running your SAS program. After all, changing a format in a table or making some changes to a graph's appearance doesn't change any of the underlying data; it is purely a cosmetic change.

Returning to the storyline, now when your boss asks you for your previous results in a different format, you can head directly to your ODS document. Using the ODS HTML destination with the CSSSTYLE= option set to the latest company style sheet, you invoke the DOCUMENT procedure. You re-create all of your original results no matter whether they were produced two weeks ago or two years ago. Furthermore, the output objects in an ODS document can be labeled and dated so you know you have the correct output without the guesswork.

Your boss arrives at your cubicle and sees the desired results already on your screen. There's a smile on your face, a smile on your boss's face, and everyone lives happily ever after.

Why Use an ODS Document?

As a project grows, the ODS Document Facility provides you with a method to conveniently store SAS procedure output, graphs, and data sets all in one location, the ODS document.

As suggested in the second scenario, here are some of the things that you can do with the ODS Document Facility:

- Replay output in an order different from that with which it was originally stored
- Search through and filter an ODS document by date, description, or output type
- Instantly turn these results into professionally styled ODS output
- Add notes about your output
- Change titles, footnotes, and create additional annotative comments

And of course, all of this is done—with the soon-to-be-familiar refrain—without re-running the original code that produced the output.

The ODS Document Facility is an excellent tool when you want to provide output to external clients, vendors, third parties, etc., who must be assured that your results are stored in the document as they were when first produced. This is particularly important for auditing or regulatory submission in the financial and pharmaceutical industries.

Why Not Manage Change with Office Software?

You can also manage your output by cutting and pasting SAS output to commercial office software, such as Microsoft Office or Open Office. If you choose to manipulate output using the third-party software to modify the appearance of your results, you may create versioning problems between your delivered work and your SAS output. If you use the ODS Document Facility, however, you can date and label your output, keep track of multiple versions, and search for output meeting particular criteria, all within the familiar SAS environment.

QuickStart Guide

By now, you should be convinced of the value of the DOCUMENT procedure and be ready to start using it. The following series of examples shows you how to get started with the ODS Document Facility. You will learn how to create a document, replay the document, as well as create a table of contents and customize the order in which the output appears in a document.

Attempting to cover a lot of ground with a few well-chosen examples does impose some limitations. In order to keep the discussions short, some exceptions to the rules and some subtleties are deferred to the appropriate chapter.

File Location

Since individual file systems differ, filenames in this quickstart guide do not include a directory name in the file path. To make the files easier to find, save the files to a directory of your own choosing, and include this directory in your file path. You will see this written in the quickstart examples:

```
...file "cars.html"
```

When you see that, write instead:

```
file "your-favorite-path-for-saving-files\cars.html";
```

Also, you should define a permanent library named MYLIB in which to store your ODS documents and make sure this definition persists between sessions.

Data

The data set that is used is SASHELP.CARS, excluding the three observations that contain TYPE='Hybrid.'

Program 1.1: Setting Up the Data

```
libname mylib "mylib-library-directory-name";
proc sort data=sashelp.cars out=WORK.cars;
  /* nothing against Hybrids, I promise! */
  where upcase(type) ne 'HYBRID';
  by origin type;
run;
```

Saving Output to an ODS Document

Saving documents is as simple as putting the ODS DOCUMENT NAME= statement before your code and the ODS DOCUMENT CLOSE statement at the end. The argument that follows the NAME= is the name of the document in which you plan to save your output. You must always provide this.

Program 1.2 assumes that there is not already a document named `mylib.quickstart`. If there is, change the definition of the MYLIB library so that you do not overwrite or mess up any existing work.

The ODS DOCUMENT statement in Program 1.2 is an example of opening an *ODS destination*. An ODS destination typically receives output and produces a version of that output in a particular file format. This format is then displayed with another application, such as a web browser, PDF reader, word processor, or other program capable of reading the output file. When you send output to an ODS document, however, the document is not displayed, but saved in a special file called a SAS *item store*. Instead of reading the output file with a third-party application, you read it with the DOCUMENT procedure.

Program 1.2: Saving Output to an ODS Document

```
ods document name=mylib.quickstart;
proc contents data=cars;
run;
proc univariate data=cars;
by origin type;
run;
proc means nmiss mean max min data=cars;
class origin type;
run;
ods document close;
```

If you want to send additional output to the document, you run another block of code using the same technique as shown in Program 1.3. The new output will be placed at the end, after existing output.

Program 1.3: Adding Additional Output to an ODS Document

```
ods document name=mylib.quickstart;
proc freq data=sasuser.cars;
table origin * type / list;
run;
ods document close;
```

Note that this program adds the output from the FREQ procedure to the end of the document. What if you wanted to add it to the beginning of the document? You can move folders to any position within a document with the MOVE TO statement, which will be discussed in Chapter 6, “Managing Folders.”

At this point, you should have a document with output from the CONTENTS, UNIVARIATE, and FREQ procedures. The next step is to use the DOCUMENT procedure to send the output to another destination. This is called *replaying*.

Replaying Output

Program 1.4 represents the shortest, yet most useful example of PROC DOCUMENT usage. All the DOCUMENT procedure really comes down to is this basic syntax:

```
proc document name=replay;run;quit;
```

The rest is just details. In this section, remember that all calls to the DOCUMENT procedure must include the NAME= argument.

Remember to Use RUN Statements!

Another important aspect of using the DOCUMENT procedure should be brought up before you start: The RUN statement is very important. Furthermore, like other interactive procedures such as the SQL procedure and the REG procedure, the DOCUMENT procedure must be terminated by a QUIT statement.

Program 1.4: Replying the Output to the HTML Destination

```
ods html file="program1_4.html";
proc document name=mylib.quickstart;
  replay;
  run;
  quit;
ods html close;
```

The output is shown as Output 1.1. Once your output is saved to the document, you can print it out as many times as you want, in whatever form you want.

Output 1.1: Replying Output as HTML

| The UNIVARIATE Procedure | | | |
|-----------------------------------|------------|-------------------------|------------|
| Variable: MSRP | | | |
| Origin=Asia Type=SUV | | | |
| Moments | | | |
| N | 25 | Sum Weights | 25 |
| Mean | 29569 | Sum Observations | 739225 |
| Std Deviation | 11842.546 | Variance | 140245895 |
| Skewness | 1.51941545 | Kurtosis | 2.366336 |
| Uncorrected SS | 2.5224E10 | Corrected SS | 3365901476 |
| Coeff Variation | 40.050546 | Std Error Mean | 2368.50919 |
| Basic Statistical Measures | | | |
| Location | | Variability | |
| Mean | 29569.00 | Std Deviation | 11843 |
| Median | 27560.00 | Variance | 140245895 |

Program 1.5 sends the same output as Program 1.4, but to a PDF file.

Program 1.5: Replying the Output to a PDF File

```

ods pdf file="program1_5.pdf";
proc document name=mylib.quickstart;
  replay;
  run;
quit;
ods pdf close;

```

Output 1.2 shows the same output, but in PDF. Notice how you didn't have to re-run your original code (Program 1.1) to create a presentation of this same output in a different file format.

Output 1.2: PDF Output Excerpt

| The UNIVARIATE Procedure | | | |
|-----------------------------------|------------|---------------------|------------------|
| Variable: MSRP | | | |
| Origin=Asia Type=SUV | | | |
| Moments | | | |
| N | 25 | Sum Weights | 25 |
| Mean | 29569 | Sum Observations | 739225 |
| Std Deviation | 11842.546 | Variance | 140245895 |
| Skewness | 1.51941545 | Kurtosis | 2.366336 |
| Uncorrected SS | 2.5224E10 | Corrected SS | 3365901476 |
| Coeff Variation | 40.050546 | Std Error Mean | 2368.50919 |
| Basic Statistical Measures | | | |
| Location | | Variability | |
| Mean | 29569.00 | Std Deviation | 11843 |
| Median | 27560.00 | Variance | 140245895 |
| Mode | . | Range | 47637 |
| | | Interquartile Range | 13391 |
| Tests for Location: Mu0=0 | | | |
| Test | | Statistic | p Value |
| Student's t | t | 12.49422 | Pr > t <.0001 |
| Sign | M | 12.5 | Pr >= M <.0001 |
| Signed Rank | S | 162.5 | Pr >= S <.0001 |

You can also replay your output in a different style, as demonstrated by Program 1.6. A complete list of styles that are shipped with SAS 9.3 is given in the appendix. You can also write your own styles, and they will work with the REPLAY statement, but writing custom styles is beyond the scope of this book.

Program 1.6: Replying the Output in a Different Style

```

ods html file="fancy.html" style=rsvp;
proc document name=mylib.quickstart;
  replay;
  run;
quit;

```

With the exception of being in a different font, with different background colors, this is the same output as Program 1.5, so it is not illustrated. Printing the same output to different destinations and different styles is *the raison-d'être* of the DOCUMENT procedure.

Subsetting Output

Being able to replay your output in different formats is powerful, but the amount of output on even a medium-sized project might grow to be quite large. You need a way to replay only what you need. Program 1.7 shows how to replay only the output from the UNIVARIATE procedure. Program 1.8 demonstrates a simple version of the WHERE= clause to exclude output from your presentations. WHERE= clauses are covered in more detail in Chapter 4, “Listing Documents Using the DOCUMENT Procedure,” as well as in Chapter 5, “The REPLAY Statement.”

Output is stored in the document by each SAS procedure. Each procedure is stored in a folder named for the procedure. If there is one folder per procedure, and there is one name per folder, you might wonder if you can have more than one folder with the same procedure name in the same document. SAS will keep them separate for you through an automatic *sequence numbering* system. The characters #1 after `univariate` refer to the first folder named `univariate`. In this case, it is the only one. If there were more, you would refer to those outputs as `univariate#2`, `univariate#3`, and so on.

Program 1.7: Restricting the Replay to Include Only UNIVARIATE Procedure Output

```
ods html file="program1_7.html" style=statdoc;
proc document name=mylib.quickstart;
  replay univariate#1;
  run;
  quit;
ods html close;
```

Not only can you restrict your output to a particular procedure, but you can also remove specific tables from your output. In Program 1.8, the Extreme Observations tables are not displayed in the replay of the document. They were excluded because, although useful to SAS programmers, they might not be useful in a report that is presented to clients.

Subsetting output is done through WHERE= options, which work the same way as they do for the WHERE= options in the DATA step. One way among several to subset output from the DOCUMENT procedure is to use special document variables. This quick-start guide will discuss one such variable. You can search the document based on the value of `_labelpath_`, one of many special document variables. These special variables are covered in Chapter 4. The characters !? mean *does not contain*.

Program 1.8: Excluding Extreme Observation Output

```
ods html file="program1_8.html" style=statdoc;
proc document name=mylib.quickstart;
  replay univariate(where=(_labelpath_ !? 'Extreme Observations'));
  run;
  quit;
ods html close;
```

The output for Program 1.8 is the same as Output 1.7, except that the Extreme Observations tables are not present. If you created output using BY-group variables, you can also use those variables to subset output, as shown in Program 1.9. BY-group variables do not require the underscore, since they are ordinary data set variables.

Program 1.9: Replying Output Using BY-Group Variables

```
ods html file="program1_9.html" style=statdoc;
proc document name=mylib.quickstart;
  replay univariate(where=(origin='Asia'));
  run;
  quit;
```

ods html close;

As with the standard WHERE= option, clauses can be combined using logical operators such as AND, OR, and NOT. Program 1.10 presents one last version of WHERE options, and brings in one more special document variable: the _Name_ variable. Every ODS object has a name, and you will learn more about how to remember the names of your output in Chapter 2, “The ODS DOCUMENT Destination.” The two objects that are the focus of this quick-start guide are ExtremeObs from Program 1.8, and Moments, which is demonstrated in Program 1.10.

The ? operator means *contains*.

Program 1.10: Subsetting with Multiple Conditions

```
ods html file="program1_10.html" style=htmlblue;
proc document name=mylib.quickstart;
  replay univariate(where=(_labelpath_ ? 'MPG' and _name_ = 'Moments' and
    type='SUV'));
  run;
  quit;
ods html close;
```

This is a good place to step back and see what you've accomplished with the DOCUMENT procedure. Output 1.3 consists of several versions of the same table, the moments table. Because all of the tables have the same structure, it is easy to review them as a group. It's very powerful to have the ability to save output in the SAS default order, yet replay any selection of that output based on what is important to you. This is quite a powerful feature of the DOCUMENT procedure. By using subsetting with WHERE= statements, you can now create outputs that would be quite challenging to produce without the DOCUMENT procedure. Finally, since the original output stored in the document has not changed, you can replay it in a different order next time.

Output 1.3 shows the output from the UNIVARIATE procedure, but only for MPG_HIGHWAY and MPG_CITY, and only the moments table.

Output 1.3: Highway and City Gas Mileage Output

| The SAS System | | | |
|---------------------------------|------------|------------------|------------|
| The UNIVARIATE Procedure | | | |
| Variable: MPG_City (MPG (City)) | | | |
| Origin=Asia Type=SUV | | | |
| Moments | | | |
| N | 25 | Sum Weights | 25 |
| Mean | 17.32 | Sum Observations | 433 |
| Std Deviation | 2.76465791 | Variance | 7.64333333 |
| Skewness | 0.12662092 | Kurtosis | -0.8227381 |
| Uncorrected SS | 7683 | Corrected SS | 183.44 |
| Coeff Variation | 15.9622281 | Std Error Mean | 0.55293158 |

| The SAS System | | | |
|---------------------------------------|------------|------------------|------------|
| The UNIVARIATE Procedure | | | |
| Variable: MPG_Highway (MPG (Highway)) | | | |
| Origin=Asia Type=SUV | | | |
| Moments | | | |
| N | 25 | Sum Weights | 25 |
| Mean | 21.68 | Sum Observations | 542 |
| Std Deviation | 3.00998339 | Variance | 9.06 |
| Skewness | 0.16534859 | Kurtosis | -0.8422186 |
| Uncorrected SS | 11968 | Corrected SS | 217.44 |
| Coeff Variation | 13.8836872 | Std Error Mean | 0.60199668 |

Managing Output

Although the primary function of the DOCUMENT procedure is replaying output without re-running the original analysis, another important function that it provides is the ability to manage your output. This section of the quickstart guide begins with learning a document's contents, and displaying a table of contents using the LIST statement.

Without using any options, the LIST statement displays a list of all the folders at the top level of the directory tree. This is useful for a quick overview of the ODS document's contents.

Program 1.11: Creating a Summary Table of Contents

```
proc document name=mylib.quickstart  
list; /* one line for each proc */  
run;  
quit
```

In Output 1.4 there is a short listing of the document, with one line for each procedure stored. The sequence numbers are placed automatically.

Output 1.4: High Level Overview of the Document

| Listing of: \Mylib.Quickstart\ | | |
|--------------------------------|--------------|------|
| Order by: Insertion | | |
| Number of levels: 1 | | |
| Obs | Path | Type |
| 1 | Contents#1 | Dir |
| 2 | Univariate#1 | Dir |
| 3 | Means#1 | Dir |
| 4 | Freq#1 | Dir |

The ODS document item store actually contains a complete directory structure, and there are multiple levels of directories in a typical one. To see the entire directory tree, use the LEVELS=ALL option, which shows all levels down to the procedure output level. The DETAILS option shows additional fields, such as when the output was created. For more information about the fields shown in the DETAILS statement, see Chapter 4, “Listing the Documents Using the DOCUMENT Procedure” as well as Chapter 10, “Working with Links” and Chapter 11, “Working with Templates.”

Program 1.12: Creating a More Detailed Listing

```
proc document name=mylib.quickstart;  
list / levels=all details; /* every subdirectory */  
run;  
quit;
```

Output 1.5 shows the results. Here are a few features worth noting. The listing shows every element that is stored in the document. The actual output that you would see if you replayed the document is listed as type TABLE. TABLE is one of several types of *output objects* that can be

stored in the document. Each entry for a table also shows its size and its full path. The list also shows every directory and subdirectory, along with a descriptive label. If you look at the full path for an output object, such as observation #10, the last entry in the path is the *ODS name* of the object. You can see the familiar `moments` object along with the `ExtremeObs` that were featured in Program 1.10 and Program 1.8, respectively.

Output 1.5: The Detailed Listing with All Levels Shown

| Listing of: \Mylib.Quickstart | | | | |
|-------------------------------|--|-------|---------------|--|
| Order by: Insertion | | | | |
| Number of levels: All | | | | |
| Obs | Path | Type | Size in Bytes | Label |
| 1 | \Contents#1 | Dir | | The Contents Procedure |
| 2 | \Contents#1\DataSet#1 | Dir | | WORK.CARS |
| 3 | \Contents#1\DataSet#1\Attributes#1 | Table | 716 | Attributes |
| 4 | \Contents#1\DataSet#1\EngineHost#1 | Table | 563 | Engine/Host Information |
| 5 | \Contents#1\DataSet#1\Variables#1 | Table | 557 | Variables |
| 6 | \Contents#1\DataSet#1\Sortedby#1 | Table | 309 | Sortedby |
| 7 | \Univariate#1 | Dir | | The Univariate Procedure |
| 8 | \Univariate#1\ByGroup1#1 | Dir | | Origin=Asia Type=SUV |
| 9 | \Univariate#1\ByGroup1#1\MSRP#1 | Dir | | MSRP |
| 10 | \Univariate#1\ByGroup1#1\MSRP#1\Moments#1 | Table | 613 | Moments |
| 11 | \Univariate#1\ByGroup1#1\MSRP#1\BasicMeasures#1 | Table | 522 | Basic Measures of Location and Variability |
| 12 | \Univariate#1\ByGroup1#1\MSRP#1\TestsForLocation#1 | Table | 574 | Tests For Location |
| 13 | \Univariate#1\ByGroup1#1\MSRP#1\Quantiles#1 | Table | 597 | Quantiles |
| 14 | \Univariate#1\ByGroup1#1\MSRP#1\ExtremeObs#1 | Table | 408 | Extreme Observations |

Changing Titles and Footnotes

If you notice in the detailed listing, you see directories and output objects. Output objects include TABLE and GRAPH objects, although there are other kinds as well. Everything that is discussed for these two types of output carries over to the others. For the record, there are also the following types of output that can be stored in a document.

- REPORT
- CROSSTAB
- EQUATION

Output objects have their own titles and footnotes that can be changed with the OBTITLE and OBFOOTN statement. Program 1.13 shows how to change a title. Once changed, that title stays stored with the document until it is changed again. When you replay, the title that is stored with the object usually takes precedence over the active SAS title. (This behavior can be overruled. This is covered in Chapter 5, “The REPLAY Statement.” See the discussion on the ACTIVETITLE option.)

Program 1.13: Changing a Title and Replaying It

```
proc document name=mylib.quickstart;
  obtitle 'Univariate#1\By Group 1#1\MSRP#1\Moments#1 'Manufacturer's
           Suggested Retail Price';
  replay 'Univariate#1\By Group 1#1\MSRP#1\Moments#1';
  run;
  quit;
```

In this example, in order to get the object name into the statement, the author cut and pasted the pathname directly from the output window. Seeing how output objects can be rather long, it won't surprise you to know that there are easier and shorter ways to specify output objects. However, cutting and pasting is reasonable if you are changing only one or two output objects at a time. Some of these other methods will be discussed in Chapter 6, “Managing Folders” and Chapter 7, “Customizing Output.” Output 1.6 shows the output with the new title in place.

Output 1.6: Changing a Title of an Output Object

Manufacturer's Suggested Retail Price

The UNIVARIATE Procedure
Variable: MSRP

Origin=Asia Type=SUV

| Moments | | | |
|---------|-------|------------------|--------|
| N | 25 | Sum Weights | 25 |
| Mean | 20550 | Sum Observations | 720225 |

Using Folders

You can create your own folders and subfolders using the MAKE statement, and then instruct the DOCUMENT destination to save your output to your new directory. Program 1.14 shows the creation of a new directory, and saving output to it.

Program 1.14: Saving Output to a User-created Directory

```
proc document name=mylib.quickstart;
make summaries;
run;
quit;

ods document name=mylib.quickstart dir=(path=summaries);
proc report nowd data=cars;
  column origin type msrp;
  define origin / group order=data;
  define type / 'Veh. Type' group order=data;
  define msrp / analysis mean;
run;
ods document close;
/* now open the document for viewing */
proc document name=mylib.quickstart;
  list summaries; /* omitting sequence no defaults to highest */
run;
quit;
```

Output 1.7 shows how the document now appears, with the summaries#1 directory and the output for the REPORT procedure inside that. There are three levels to this directory, and the actual table is at the third level. Sometimes you need to look carefully at the document produced so that you can find your output. It might be structured differently than you expect at first.

Output 1.7: Showing Folder for REPORT Procedure Nested in Summaries#1

| Listing of: \Mylib.Quickstart\summaries#1 | | | | |
|---|---|-------|------------------|-----------------------------------|
| Order by: Insertion | | | | |
| Number of levels: All | | | | |
| Obs | Path | Type | Size in Bytes | Label |
| 1 | \summaries#1\Report#1 | Dir | | The Report Procedure |
| 2 | \summaries#1\Report#1\Report#1 | Dir | | Detailed and/or summarized report |
| 3 | \summaries#1\Report#1\Report#1\Report#1 | Table | 1406 | |

If you want to replay the report, and only the report, any of the following three commands shown in Program 1.15 would help. Program 1.15 also features several statements used in the same DOCUMENT procedure call. Each is separated by a RUN statement. The second REPLAY statement omits the sequence number #1. If omitted, the sequence number will default to the most recent version of the output.

Program 1.15 : Replaying the Report

```
proc document name=mylib.quickstart;
replay summaries#1;
run;
replay Summaries#1\Report; /* Report becomes Report#1 */
run;
replay Summaries#1\Report#1\Report#1;
run;
quit;
```

When you are searching on the *value* of the path, the search is case sensitive. However, the pathname itself is case insensitive.

Thus, the following statement will still replay correctly.

```
replay summaries#1\report#1\report#1;  
run;
```

By contrast, when you compare the `_PATH_` variable to a string constant, as in Program 1.10, the string must match exactly, including case.

Deleting and Overwriting Document Contents

Program 1.16 shows how to clear an entire document. The new idea in this code is the *WRITE access mode*. The default behavior of the document destination is to open documents in UPDATE access mode. This causes new output to be placed into the document after any existing output. The WRITE access mode opens a document and clears its contents before writing.

Program 1.16: Starting Over

```
ods document name=mylib.quickstart(write);
<different, newer sas code>
ods document close;
```

The document `mylib.quickstart` will now have only the results of the body of the new SAS code. The original analysis from Program 1.1 will no longer be retrievable.

Program 1.17 demonstrates how to delete documents with the DOCUMENT procedure. The DELETE statement does not give you any second chances, so be sure you want to delete before you do so. Nonetheless, sometimes a document can grow to become unwieldy, and deleting and starting over might be the best choice, as in Program 1.16. Still, there are usually better alternatives to deleting your hard work, and Chapter 6, “Managing Folders” discusses some of these alternatives, such as archiving unwanted work in another document or folder.

When invoking the DOCUMENT procedure, you must have a current document, even when you are using the DELETE command to delete the current document, as in Program 1.17.

Program 1.17: Deleting a Document

```
proc document name=dontdoit;
  delete dontdoit;
run;
quit;
```

What Remains to Be Covered?

Now that you have learned to create a document, store output, and replay the output, you might be wondering what is left to learn. The remaining chapters will show you how to refine the basic skills presented in the quick-start guide, such as:

- Learning more ways to conditionally subset when replaying part of a document
- Using the MOVE TO statement with positioning options to change the order in which output is to be replayed
- Storing more than one procedure of the same name in a document
- Labeling your output so that you know the purpose for a collection of output as well as its contents
- Changing titles and footnotes before replaying the document
- Adding text notes to your work that appears when you replay
- Making many modifications by changing the ODS Template that was used to create the original output.

Summary

The ODS Document Facility maintains an ongoing journal of your output that you can replay at any time without re-running the original analysis. This approach has several advantages. By delaying the tasks of printing and changing layout and appearance, you can focus on your main work. When it comes time to present the output, you can experiment with a variety of destinations, output formats, and style options, all without making any changes to or re-running your original code.

Chapter 2: The ODS DOCUMENT Destination

[Introduction](#)

[Convention](#)

[What Is the Output Delivery System \(ODS\)?](#)

[Definition of the ODS Destination](#)

[ODS without the Document Facility](#)

[ODS with the DOCUMENT Destination](#)

[Role of the DOCUMENT Procedure](#)

[Reasons to Use the Document](#)

[Naming ODS Output](#)

[The ODS TRACE Statement](#)

[Determining the Directory Where ODS Output Is Stored](#)

[Implementation Details of the Item Store](#)

[Controlling Output Using SELECT and EXCLUDE Lists](#)

[Summary](#)

Introduction

The quick-start guide in Chapter 1 provided an overview of the entire process of storing and replaying ODS output. In the following chapters, the goal is to go into more detail about each part of the process, in order to help you use the features of the ODS Document Facility more intelligently. This chapter answers the following questions:

- What is an ODS destination?
- What is the role of the DOCUMENT destination in ODS?
- How does the DOCUMENT destination store your output?
- What else is stored in the DOCUMENT destination?
- How can you control what SAS procedure output is stored in a document?

The chapter gives a very brief overview of the main parts of ODS and how it functions. It also describes the special role that the ODS DOCUMENT destination plays as a midway point to keep SAS procedure output ready until you need to print it.

Convention

When discussing any ODS destination, this book uses all caps, as in *OUTPUT destination* or *DOCUMENT destination*. When talking about a particular file that actually holds the data, the convention is to use a common noun, as in *the document*.

This chapter, as well as the rest of the book, refers to *SAS procedure output*. You might already be familiar with ODS in the DATA step. The book uses the phrase SAS procedure output to refer to that type of output as well. In contexts in which no confusion arises, the book sometimes refers simply to *output*.

What Is the Output Delivery System (ODS)?

Before asking the question “What is a Destination?” you need a brief overview of the Output Delivery System (ODS). ODS was created in order to display SAS output in a variety of file formats, including HTML, RTF, and PDF as well as the traditional monospace output.

Definition of the ODS Destination

You can think of an ODS destination as a program that turns SAS output into printable output. The term *printable output* means a file that a user can read outside SAS by using a familiar application. This familiar application can be a Web browser, a spreadsheet, a PDF reader, or something else.

Some texts also refer to the file as the *destination*. This author prefers to think of destinations as *programs* that produce printable output. The reasoning here: programs are configurable with options, and can do things, whereas a file is just a set of bits on a storage device.

There are a number of ODS destinations, or *destinations* for short, to meet a variety of communication needs. Following are descriptions of some of the more common destinations.

- HTML: This destination produces output in Hypertext Markup Language (HTML), which is amenable for viewing in a Web browser. In SAS 9.3, it is the default destination for the SAS windowing environment.
- RTF: This destination creates output in Rich Text Format, which can be viewed in Microsoft Word or Microsoft WordPad.
- PDF: This destination produces output in the Portable Document Format (PDF), which can be viewed using Adobe Acrobat Reader.
- OUTPUT: This destination sends output to SAS data sets. Like the DOCUMENT destination, this is a destination that is controlled by SAS. You could say that SAS itself is the viewer for this destination's contents.
- LISTING: This destination produces output in monospace (Courier-type) font. LISTING output is still available, but as of SAS 9.3, the default output destination in a windowing environment is HTML. This means newer SAS users might go through their entire career never seeing SAS LISTING output, but it is still provided for backward compatibility.

There are many more ODS destinations, with new ones in development all the time.

ODS without the Document Facility

To get a sense of the role of the ODS Document Facility within ODS, you first need a basic knowledge of how ODS works without the Document Facility. This is illustrated through the following example program.

Creating output with ODS consists of three steps: opening one or more destinations, using SAS procedures to create output, and closing the destination. Closing the destination triggers the destination to convert its copy of the procedure output to a final output form. Final output allows users who do not have SAS to view the file. In Program 2.1, this output takes the form of a PDF file and an HTML file.

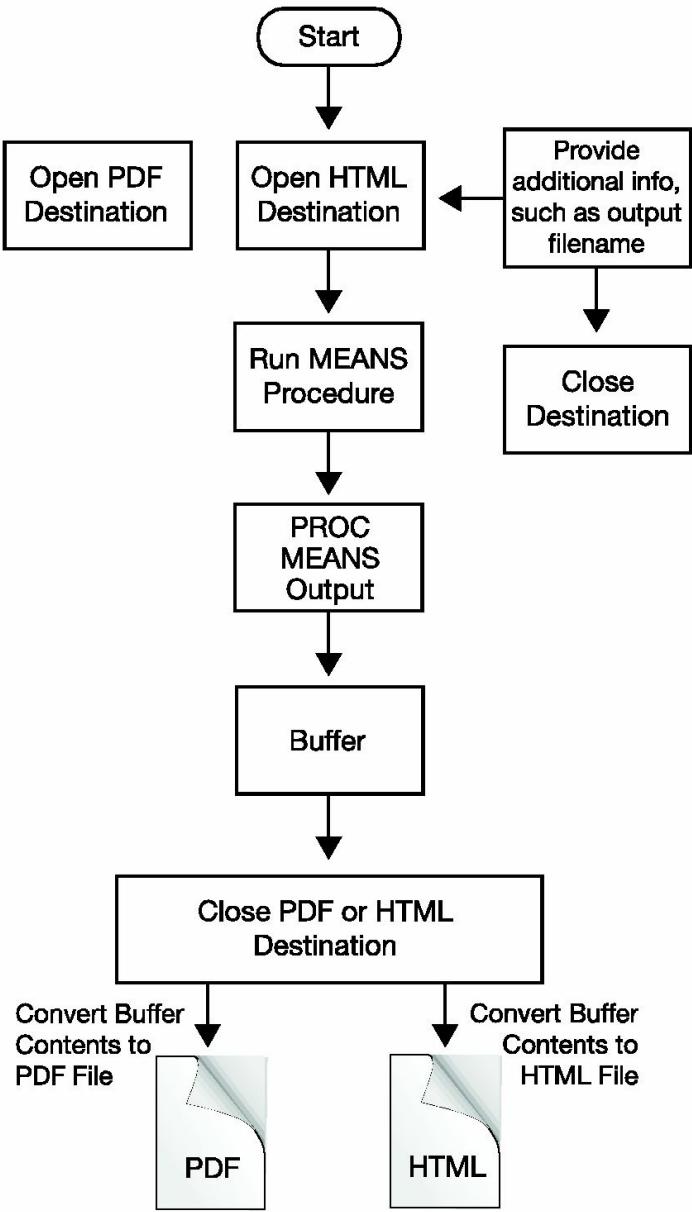
Although Program 2.1 is very simple, it does illustrate several important aspects that all ODS code must have. In addition, the description of how Program 2.1 works is meant as a model. It is not a technically precise description of how ODS works, but it should be sufficient to help you understand how the ODS Document Facility interacts with the rest of ODS.

Program 2.1: Sending SAS Output to an ODS Destination

```
ods html file="curious.html"; ①  
ods pdf file="curious.pdf";  
title "MPG";  
proc means data=sashelp.cars; ②  
  var mpg_highway;  
run; ③  
ods pdf close;  
ods html close; ④
```

- ① The first two lines open two destinations. You can have as many open as you want to have. Two will suffice for the purpose of this demonstration. The destinations are programs that will convert the MEANS procedure output to an external format, once they get permission to proceed. For now, the destinations will wait until they get the signal to continue.
- ② Meanwhile, the MEANS procedure produces tabular (and depending on the additional parameters supplied, possibly graphical) output, which ODS stores in an intermediate form as tabular data. For the purpose of this example, please accept the simplification that all ODS output, even graphical output, is represented as tabular data.
- ③ The RUN statement is critically important. It signals to ODS to bundle up the previous MEANS statement, execute it, and send the output to a file area. At this point, this secret buffer contains what will be called preformatted output. This is output that has all the information another destination would need to create an external file. This information would include the data table itself and instructions for formatting it.
- ④ The CLOSE statement closes the file. At this point, the PDF and HTML files convert their output stored in memory to their final format and send the result to a disk file. Also, the files become visible to the operating system. Before you close the files, the pending output is just sitting in memory, waiting. In short, until the destinations are closed, nothing happens. Procedure output from other programs continues to accumulate in the destination's buffer area until the destination is closed.

Figure 2.1: An Overview of Program 2.1 - Sending Output to ODS Destinations



ODS with the DOCUMENT Destination

The ODS DOCUMENT destination takes the process in Figure 2.1 and splits it in half. Now that you understand that destinations are responsible for taking SAS output and converting it to an external format, it's not a large jump to see how the document works.

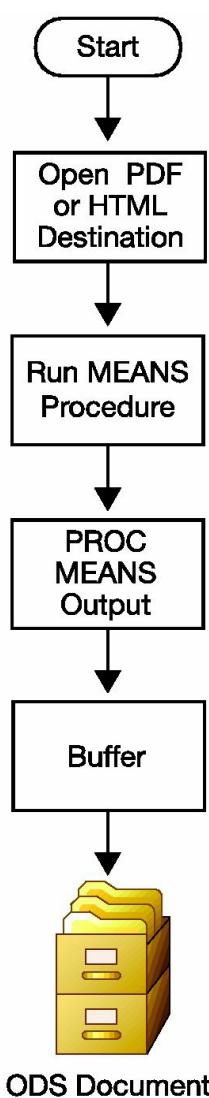
When output is sent to the ODS DOCUMENT destination, the first part works the same as illustrated in Figure 2.1. ODS still assembles the information that it needs to produce output, but instead of running through the conversion process, ODS stores the preformatted output as an *item store*. See Figure 2.2, where the item store is represented as a file cabinet.

Unlike other destinations, which produce a new file each time you send output to them, this destination enables you to continue to add output cumulatively to the same document over the course of several programs. The DOCUMENT destination is the only destination that has this property. Although all destinations accumulate output until you close them, only the DOCUMENT destination continues to accumulate output even after you close and subsequently reopen it. It appends the new output to the output that is already stored.

In Chapter 3, “The ODS DOCUMENT Statement,” you will learn how to modify this behavior, but this description is a correct description of the DOCUMENT destination’s default behavior.

This capability of accumulating procedure output from several programs and different sessions is vital. It is what makes the document useful for keeping SAS procedure output easy to find and replay.

Figure 2.2: The DOCUMENT Destination Stores Preformatted ODS Output



ODS Document

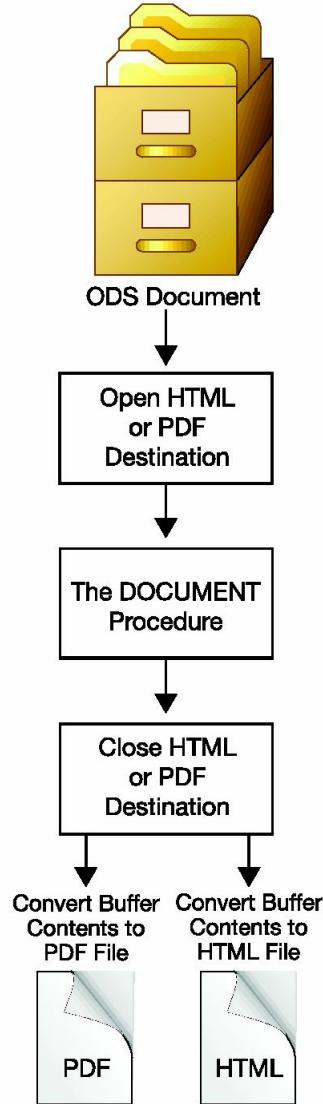
With SAS output stored in the item store in its preformatted state, ODS is now prepared to send your output to any other destination.

Role of the DOCUMENT Procedure

Figure 2.3 shows the role of the DOCUMENT procedure. It picks up where Figure 2.2 leaves off. Pulling preformatted output (the folders) from the item store, the DOCUMENT procedure sends these files directly to the destinations. The DOCUMENT procedure does not do any analysis; it simply hands procedure output over to the other destinations.

The role of producing ODS output, played by the MEANS procedure in Figure 2.1, is now played by the DOCUMENT procedure. As in Figure 2.1, when you close the destinations, the destination converts the ODS output into a file. The replay is now complete.

Figure 2.3: ODS with the Document - Printing Out to a Destination



You are not limited in the destinations that you had originally envisioned when you originally wrote the code. Once you store the output in the document, you can replay it to as many formats as you want, whenever you want.

Reasons to Use the Document

The benefit of adding this intermediate step of storing output in the document is the additional flexibility that it offers. By storing ODS output for later replay, you can adopt the philosophy of “save everything now, and worry about the details of actual output production later.” This approach will let you come up to speed on ODS quickly. Store output in the document now, and you can always come back to replay when you have found the right format for your work. With the document, you can experiment with different destinations, formats, and styles until you find the one that suits you. Furthermore, you can do this as many times as needed without running the original analysis. Your output will be there waiting for you.

Naming ODS Output

In order to start communicating with SAS about the documents that you create, you need to know a few details about how SAS procedure output is stored within the document. In particular, you need to know how ODS names this output. This is important because there is a direct correspondence between this ODS naming convention and the location in the document where your output will be stored.

Every element of output that is created by a SAS procedure and sent to an ODS destination must be given a name. Each output object is given an ODS *path*. An element of output refers to tables, graphs, crosstabs, equations, and reports. This book refers to these objects as *output objects*.

This path consists of multiple components separated by periods. The number of components depends on the options that you choose when you run the procedure. Table 2.1 shows a few examples of ODS paths.

Table 2.1: Some ODS Paths

| |
|---|
| Univariate.ByGroup2.Height.QQPplot.QQPlot |
| Freq.Table1.CrossTabFreqs |
| Tabulate.Report.Table |
| Sgpanel.sgpanel |

In order to know the pathname that ODS gives to your output, you must first understand how to find the complete ODS name of your output as ODS produces it. For this, you use the ODS TRACE statement.

The ODS TRACE Statement

The ODS TRACE statement displays to the log (or listing) all relevant information about an output object each time ODS produces one.

Here is the syntax of the ODS TRACE statement:

```
ODS Trace [ON|OFF] [/ [LABEL|EXCLUDE|LISTING] ] ;
```

Here is a description of each option:

LABEL: Shows full label paths in addition to the more succinct ODS names.

EXCLUDE: In addition to showing the output that was sent to the destination, also shows entries for output that is excluded using the ODS EXCLUDE statement. This is covered later in this chapter.

LISTING: Sends ODS TRACE output to the LISTING destination as well as to the log.

Program 2.2 shows an example of using ODS TRACE to track the output from the UNIVARIATE procedure.

Program 2.2: Demonstrating the ODS TRACE Statement

```
ods trace on / label;
ods html file="program2_2.html" style=minimal;
proc univariate data=class;
  var height age / histogram;
  qqplot age;
run;
ods html close;
ods trace off;
```

When you declare your destination, in addition to supplying an output file, you can also supply an ODS STYLE. This program creates an HTML file using the `minimal` style. Supplied by SAS, this style prints output in black and white with minimal ornamentation. A list of currently available ODS styles appears in the appendix. For more information about styles and their use, see *Output Delivery System: The Basics and Beyond*.

Output 2.1 shows the trace output for some of the tables that are produced by the program. The TRACE output is presented alongside its corresponding HTML output. The full output from Program 2.2 contains 17 separate output objects, which is far too many to illustrate in this fashion.

Let's look at the parts of the trace output in detail. As mentioned before, the path contains all the information needed to describe each output object that is sent to a destination. The far right component of the path is the ODS *name* of the object, which is also listed as the first line of the trace output. You will use the ODS name when creating SELECT and EXCLUDE lists, to be covered later in this chapter.

The first object displayed is named `moments`. The second is named `basicmeasures`. The next three objects, as they would appear in your output are `TestsForLocation`, `Quantiles`, and `ExtremeObs`. They are not shown on Output 2.1 in the interest of saving space. The far left of the pathname is the

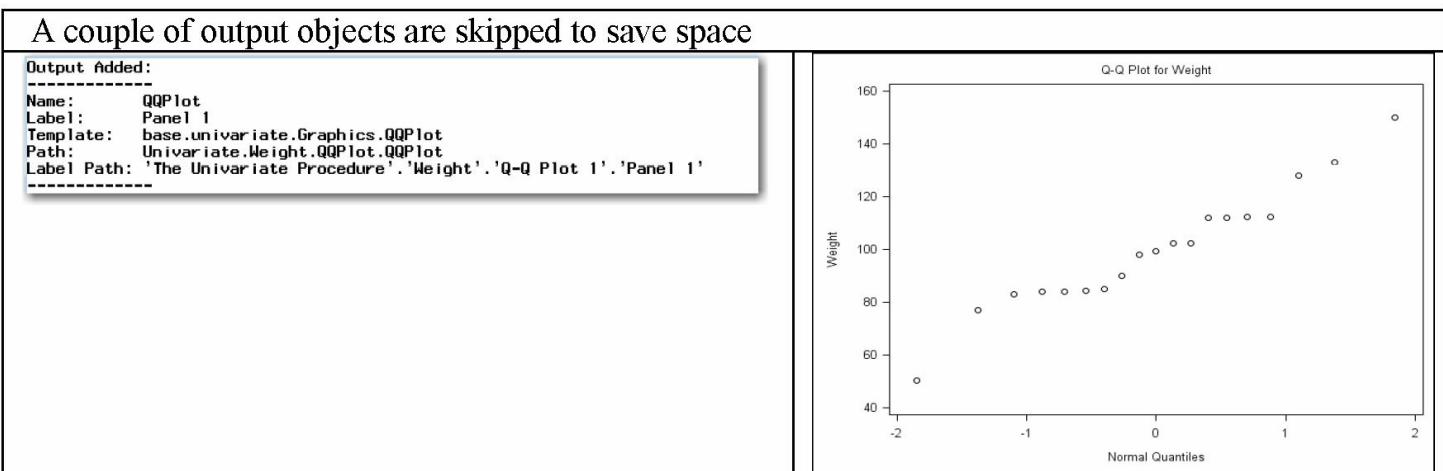
procedure name, which, in this case, is UNIVARIATE.

Finally, in between the procedure name and the ODS name might be additional levels. The number of such levels will vary depending on which BY-groups, VAR statements, and on other statements in the procedure call.

Output 2.1: SAS Procedure Output Along with the Trace of Each Output

| Trace | Output | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|--|----------------------------|------------|--|--|----------|----|-------------|----|------|------------|------------------|----------|---------------|------------|----------|------------|----------|------------|----------|------------|----------------|-----------|---------------------|------------|-----------------|------------|----------------|------------|
| <pre>Output Added: ----- Name: Moments Label: Moments Template: base.univariate.Moments Path: Univariate.Weight.Moments Label Path: 'The Univariate Procedure'.'Weight'.'Moments' -----</pre> | <p>The UNIVARIATE Procedure Variable: Weight</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="4">Moments</th> </tr> </thead> <tbody> <tr> <td>N</td><td>19</td><td>Sum Weights</td><td>19</td></tr> <tr> <td>Mean</td><td>100.026316</td><td>Sum Observations</td><td>1900.5</td></tr> <tr> <td>Std Deviation</td><td>22.7739335</td><td>Variance</td><td>518.652047</td></tr> <tr> <td>Skewness</td><td>0.18335097</td><td>Kurtosis</td><td>0.68336484</td></tr> <tr> <td>Uncorrected SS</td><td>199435.75</td><td>Corrected SS</td><td>9335.73684</td></tr> <tr> <td>Coeff Variation</td><td>22.7679419</td><td>Std Error Mean</td><td>5.22469867</td></tr> </tbody> </table> | Moments | | | | N | 19 | Sum Weights | 19 | Mean | 100.026316 | Sum Observations | 1900.5 | Std Deviation | 22.7739335 | Variance | 518.652047 | Skewness | 0.18335097 | Kurtosis | 0.68336484 | Uncorrected SS | 199435.75 | Corrected SS | 9335.73684 | Coeff Variation | 22.7679419 | Std Error Mean | 5.22469867 |
| Moments | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 19 | Sum Weights | 19 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Mean | 100.026316 | Sum Observations | 1900.5 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Std Deviation | 22.7739335 | Variance | 518.652047 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Skewness | 0.18335097 | Kurtosis | 0.68336484 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Uncorrected SS | 199435.75 | Corrected SS | 9335.73684 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Coeff Variation | 22.7679419 | Std Error Mean | 5.22469867 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <pre>Output Added: ----- Name: BasicMeasures Label: Basic Measures of Location and Variability Template: base.univariate.Measures Path: Univariate.Weight.BasicMeasures Label Path: 'The Univariate Procedure'.'Weight'.'Basic Measures of Location and Variability' -----</pre> | <p>The SAS System</p> <p>The UNIVARIATE Procedure Variable: Weight</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="4">Basic Statistical Measures</th> </tr> <tr> <th colspan="2">Location</th> <th colspan="2">Variability</th> </tr> </thead> <tbody> <tr> <td>Mean</td><td>100.0263</td><td>Std Deviation</td><td>22.77393</td></tr> <tr> <td>Median</td><td>99.5000</td><td>Variance</td><td>518.65205</td></tr> <tr> <td>Mode</td><td>84.0000</td><td>Range</td><td>99.50000</td></tr> <tr> <td></td><td></td><td>Interquartile Range</td><td>28.50000</td></tr> </tbody> </table> <p>Note: The mode displayed is the smallest of 4 modes with a count of 2.</p> | Basic Statistical Measures | | | | Location | | Variability | | Mean | 100.0263 | Std Deviation | 22.77393 | Median | 99.5000 | Variance | 518.65205 | Mode | 84.0000 | Range | 99.50000 | | | Interquartile Range | 28.50000 | | | | |
| Basic Statistical Measures | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Location | | Variability | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Mean | 100.0263 | Std Deviation | 22.77393 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Median | 99.5000 | Variance | 518.65205 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Mode | 84.0000 | Range | 99.50000 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | Interquartile Range | 28.50000 | | | | | | | | | | | | | | | | | | | | | | | | | | |

(continued)



Determining the Directory Where ODS Output Is Stored

ODS stores SAS output in a nested series of folders, based on the path. The point to remember is that every time TRACE outputs something, it is telling you “this is where I plan to store this output in the document.” It is not *necessary* to use the ODS TRACE statement when you save SAS output to an ODS destination. However, it is quite helpful to do so until you get a sense of how ODS sets the path of an output.

For example, consider from Output 2.1, the ODS output object that has the following pathname:

Univariate.Weight.Moments

This would be stored in the document at the path:

\Univariate#1\Weight#1\Moments#1

The numbers after the hash sign (#) are called *sequence numbers*. Sequence numbers exist to distinguish output that is produced from SAS programs that have been run multiple times. This topic will be covered in more detail in the next chapter.

What remains is to figure out where the folder Univariate will be stored. The location can vary, but if no further information is provided, the default is to store the folders at the top level of the document. Therefore, the moments output would be stored in the document at the following address, or path:

\Univariate#1\Weight#1\Moments#1

Implementation Details of the Item Store

Besides storing SAS output in a directory structure based on the pathname, SAS stores additional information about your output. For example, SAS must store information about any formats needed to display output, as well as a table template, if applicable. It is not necessary to know anything about table templates in order to understand the DOCUMENT destination; you need to know only that ODS must store some formatting information as part of saving SAS procedure output to the item store.

The ODS document is not portable because the underlying item store file format is operating system dependent. The term *operating system* is also stricter than you might think. For example, 64-bit Windows is considered to be a different operating system from 32-bit Windows. Thus, teammates who are running the same operating system can read from each other's documents only if they are running precisely the same operating system. However, documents are portable upward between versions. Thus, a document that was created in SAS 9.2 will still be readable in SAS 9.3.

There is one more ODS topic that does not fit easily in the other chapters, but that pertains to which data is stored in the document. If you do not want the document to store all of your SAS procedure output, the next section will be of interest.

Controlling Output Using SELECT and EXCLUDE Lists

SELECT and EXCLUDE lists are another ODS feature that is common to all destinations, and the DOCUMENT destination is no exception. This section shows how to restrict which output ODS sends to the DOCUMENT destination. In the interests of brevity, this section covers only enough information to use SELECT and EXCLUDE with the DOCUMENT destination.

You can use SELECT and EXCLUDE statements with other destinations besides the DOCUMENT destination, and you can use these for ODS as a whole. Although this chapter doesn't cover these operations, a few general hints about ODS SELECT and EXCLUDE lists are in order:

- Try to avoid using both SELECT and EXCLUDE lists in the same program. Usually, you need one or the other. While using both is not forbidden, it can be confusing. For more information about the interaction of SELECT and EXCLUDE, see SAS Help and Documentation.
- Try to avoid using both destination-specific EXCLUDE lists and global ODS EXCLUDE lists unless you know the rules that govern how they interact.

These hints are purely the opinion of this author, and you should not construe them as the "proper" use of these statements.

Here is the syntax for ODS SELECT and EXCLUDE statements.

```
ODS (destination) [SELECT|EXCLUDE](name-list);
```

The ODS SHOW command, appropriately enough, shows the status of the select and exclude situation for a destination. If you find that you are missing output, run the ODS show command to ensure that you are sending the correct output to the destination.

```
ODS (destination) SHOW;
```

The effect of the ODS SELECT statement is to add a name to a list, called the SELECT list. Each destination maintains its own SELECT list. Consider the code in Program 2.3, which manages the SELECT list for the PDF destination.

Program 2.3: Selecting Output to Be Saved to the Document

```
ods document name=sample; ①  
ods document select moments extremeobs; ②  
ods document show;  
ods document close;  
ods document name=mylib.firstdoc; ③  
ods document show;  
ods document close;
```

① You must open the destination in order to use its SELECT and EXCLUDE lists.

② This statement tells the system that the *only* objects the DOCUMENT destination will accept are

Moments and Extremeobs. This means that output from any other table name, regardless of procedure, will be *omitted* from the ODS document sample.

- ③ These settings apply only to the ODS document named sample. As the select list for the ODS document MYLIB.FIRSTDOC shows, the ODS SELECT list for the first ODS document does not affect the ODS SELECT list of the others. Thus, it is safer to use destination-specific SELECT and EXCLUDE lists than it is to use the OVERALL list.

Output 2.2: Contains the Status for the DOCUMENT Destination and OVERALL

```
18 ods document name=sample;
19 ods document select moments extremeobs;
20 ods document show;
Current DOCUMENT select list is:
1. moments
2. extremeobs
21 ods document close;
22 ods document name=mylib.firstdoc;
23 ods document show;
Current DOCUMENT select list is set to default value (ALL).
Current OVERALL select list is: ALL
24 ods document close;
```

One topic that is not covered is the interaction between SELECT and EXCLUDE. If you are interested in using both of these simultaneously, the rules for their interaction are covered in SAS Help and Documentation. One reason that this is being deemphasized is that the document, when used to its full advantage, makes it unnecessary to remember such rules.

In fact, you might find it easier to avoid SELECT and EXCLUDE, and just save everything to the document. In Chapter 5, “The REPLAY Statement,” you will learn how to choose only the output that you need using conditional logic. You will get all the benefits of subsetting without having to worry if you have excluded something that you might need later. When you use SELECT or EXCLUDE to limit what you save, there is no way to recover it unless you rerun the analysis code.

Summary

This chapter defined ODS destinations, which are programs that take SAS procedure output and create external files such as HTML or PDF files. Then you learned how the DOCUMENT destination stores output from ODS in a state that can be replayed later. The DOCUMENT destination stores this output in a file called an *item store*, and within that item store is a directory system. When it is time to replay your output, the DOCUMENT procedure finishes the suspended output process by sending SAS output to the third-party destination. The purpose of suspending ODS output in this way is to gain flexibility. You can now create many different file formats for the same output, and all without having to rerun your analysis.

The ODS DOCUMENT assigns a pathname to every output that it saves to the item store. ODS transforms the ODS pathname of the output into a storage location. To make it easier to remember the names, use the ODS TRACE statement to display the path of every output object that ODS produces.

One of the reasons for learning to use the ODS DOCUMENT destination as one of your first destinations is that you can “store first, learn later.” As you learn more commands and destinations, you can always replay the output to these new destinations as you gain more facility with ODS.

Chapter 3: The ODS DOCUMENT Statement

[Introduction](#)

[The ODS DOCUMENT Statement](#)

[The NAME= Option](#)

[Example: Storing Output at the Top Level](#)

[Comparing ODS TRACE Output to ODS Document Paths](#)

[The DIR= Option](#)

[Behavior of the Pathname](#)

[The CATALOG= Option](#)

[Using Access Options \(Write and Update\)](#)

[Example: Reviewing Your Progress](#)

[Summary](#)

Introduction

Now that you understand the concept and layout of the ODS document that was covered in the previous chapter, the next step is to talk about other ways to change the behavior of the ODS DOCUMENT destination. The goal of this chapter is to show you how to use the DOCUMENT statement to open and close the DOCUMENT destination, as well as how to direct SAS output to be stored in specific folders.

You can also use the DOCUMENT statement to change some aspects of the destination's behavior while it is still open. This chapter also discusses the crucial difference between the Write and Update access mode. Update mode is the default. Each time the DOCUMENT destination is open, it accumulates SAS output without deleting existing output. Write access mode deletes any content from the folder before sending output to it.

The ODS DOCUMENT Statement

The syntax for opening an ODS document to receive output via the ODS DOCUMENT destination is as follows:

Syntax of the ODS DOCUMENT Statement

```
ODS DOCUMENT  
NAME=<libref.>member-name <(access-option)>> |  
DIR=(<PATH=path<(access-option)> <LABEL="label">> )|  
<CATALOG=permanent-catalog | _NULL_|;  
ODS DOCUMENT action;
```

The NAME= Option

The NAME= option specifies the SAS library name and member name where the ODS document is, or will be located. The NAME option is required when opening a document. You open the DOCUMENT destination by running the ODS DOCUMENT statement with the NAME= option.

In this chapter, you will create a document called `mylib.firstdoc`.

Example: Storing Output at the Top Level

You open the ODS DOCUMENT statement with the NAME= option. Once the document is open and waiting to receive output, the first step is to store some data in it. You can use any SAS procedure code that creates output, including ODS in the DATA step. ODS in the DATA step is included in the rubric of SAS Procedure code; the system sees ODS in the DATA step as being produced by the DATASETPROCEDURE. Once the document is opened, it will insert the output into the default folder. An example will make this clearer.

The data set used for this discussion is SASHELP.CLASS sorted by SEX. Program 3.1 shows how you add output to the document.

Program 3.1: Adding Output to a Document

```
/* The sort procedure just creates data, and no ODS output. */
/* The sort can be run before or after the ODS Document
   statement.*/

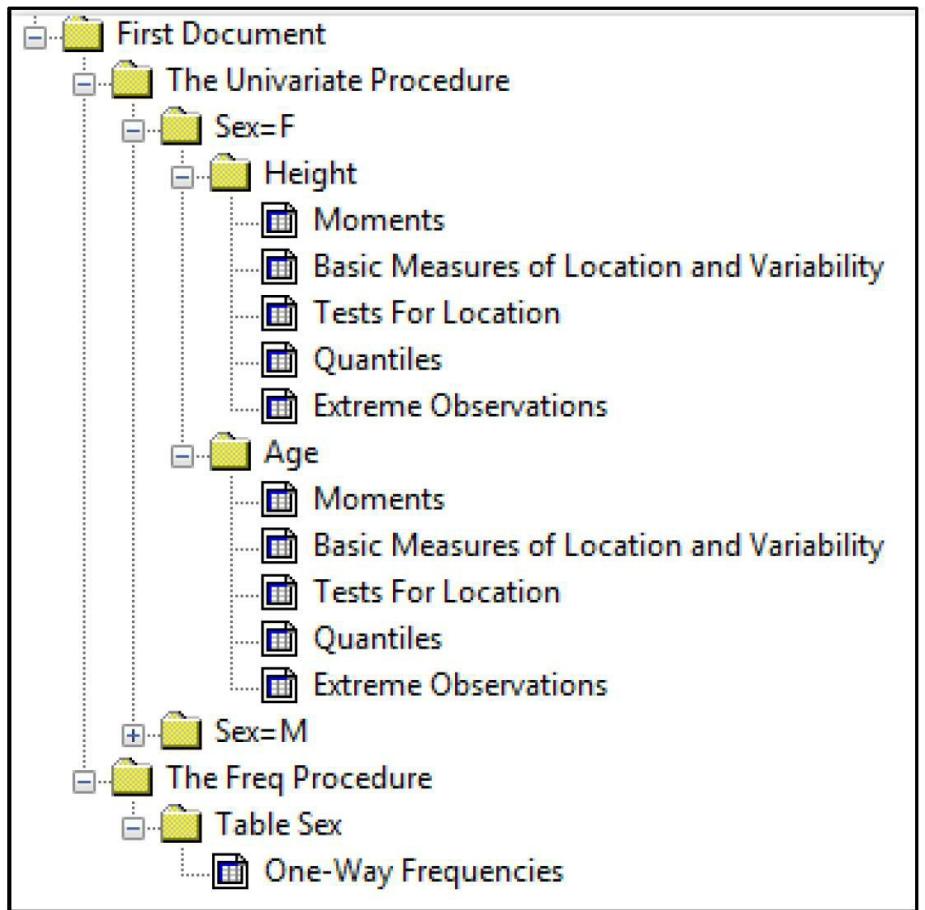
ods trace on; ①
ods document name=mylib.firstdoc;
proc sort data=sashelp.class out=class;
by sex;
run;
proc univariate data=class;
by sex;
var height age;
run;

proc freq data=class;
table sex;
run;
ods document close; ②
```

- ① As you add more output to the document, it will be helpful to see the correspondence between path locations in the ODS document and the path output by ODS. Running the UNIVARIATE and FREQ procedures adds the output from those procedures to the document. You can put any SAS procedure output into the document, as well as output from ODS in the DATA step
- ② Remember to close the document. If left open, the document will accumulate output long after you might have forgotten you opened it. A large document can create an I/O drain on the system and cause it to perform more slowly. Therefore, always close your document.

After running Program 3.1, you have a document whose output is stored in a folder system, shown in Display 3.1.

Display 3.1: View of the MYLIB.FIRSTDOC's Folder Structure



In the default behavior of the ODS document, output is added to the document in the order that it was created. The BY-group for variable SEX is stored in alphabetical order: First SEX=F, and then output for SEX=M. (To save space, the output for SEX=M is not expanded.)

Another way to see the folders and output that are stored in the document is to use the LIST statement in the DOCUMENT procedure. Although you will read the full discussion of the LIST statement in Chapter 4, “Listing Documents Using the DOCUMENT Procedure,” you have seen enough of the LIST statement in the quick-start guide, so Program 3.2 should not be too difficult to understand.

Program 3.2: Viewing the Contents of the Document

```

proc document name=mylib.firstdoc;
ods html file="program3_2.html";
list / levels=all;
run ;
quit;
ods html close;

```

Program 3.2 produces the document whose structure is shown in Output 3.1a, which shows how the output is actually arranged in an item store. It is the layout that corresponds to the conceptual picture that is shown in Display 3.1. Output 3.1a also shows how the document is laid out as nested folders, denoted by TYPE=DIR, all the way to the bottom level. At the bottom level of the folder hierarchy you can find your actual stored procedure output, denoted by TABLE or GRAPH (or possibly others, but TABLE and GRAPH will suffice for the examples in this book).

Output 3.1a: Another View of the Tree Structure of a Document

Listing of: \Mylib.Firstdoc

Order by: Insertion

Number of levels: All

| Obs | Path | Type |
|-----|--|-------|
| 1 | \Univariate#1 | Dir |
| 2 | \Univariate#1\ByGroup1#1 | Dir |
| 3 | \Univariate#1\ByGroup1#1\Height#1 | Dir |
| 4 | \Univariate#1\ByGroup1#1\Height#1\Moments#1 | Table |
| 5 | \Univariate#1\ByGroup1#1\Height#1\BasicMeasures#1 | Table |
| 6 | \Univariate#1\ByGroup1#1\Height#1\TestsForLocation#1 | Table |
| 7 | \Univariate#1\ByGroup1#1\Height#1\Quantiles#1 | Table |
| 8 | \Univariate#1\ByGroup1#1\Height#1\ExtremeObs#1 | Table |
| 9 | \Univariate#1\ByGroup1#1\Age#1 | Dir |
| 10 | \Univariate#1\ByGroup1#1\Age#1\Moments#1 | Table |
| 11 | \Univariate#1\ByGroup1#1\Age#1\BasicMeasures#1 | Table |
| 12 | \Univariate#1\ByGroup1#1\Age#1\TestsForLocation#1 | Table |
| 13 | \Univariate#1\ByGroup1#1\Age#1\Quantiles#1 | Table |
| 14 | \Univariate#1\ByGroup1#1\Age#1\ExtremeObs#1 | Table |

Comparing ODS TRACE Output to ODS Document Paths

The names for the paths are chosen based on the ODS path, which is always displayed in the log when ODS TRACE is set to ON. The trace corresponding to observation 4 (in the listing of Output 3.1a) is shown in Output 3.1b.

Output 3.1b: ODS TRACE Output

```
Output Added:  
-----  
Name: Moments  
Label: Moments  
Template: base.univariate.Moments  
Path: Univariate.ByGroup1.Height.Moments  
-----
```

Compare Output 3.1a and Output 3.1b. There is a direct correspondence between the pathname and the document path. The only difference is that the dots are changed to backslashes, and automatic sequence numbers are added. You will learn how the sequence numbering system works in Chapter 4. For the purposes of viewing your document, just notice they are part of the path. Notice that all observations in Output 3.1a have sequence numbers equal to #1.

In the default behavior, SAS output is added to the top level of the document. Each SAS procedure has its own folder, which is inserted into the document in the order that it was created. The pathname of each output is based on the ODS output path.

Now that you have saved a document, you might want to replay it to reassure yourself that the replay works. The standard code for this is Program 3.3:

Program 3.3: Making Sure the Output Is Still Present

```
proc document name=mylib.firstdoc;  
  replay;  
  run;  
  quit;
```

The output from this program is several pages long, and is not shown, but it is the same output as from Program 3.1. It is equally easy to store and replay ODS statistical graphics as it is with tabular output.

The DIR= Option

You are not limited to storing output in the top level of the document. Often, you will know in advance in which folder you would like to store your SAS output. The DIR= option makes it easy to assign SAS output to a specific folder using the DOCUMENT statement.

To refresh your memory, here is the syntax for the DIR= option:

```
DIR=(PATH=path<(access-option)> <LABEL="label"> )
```

Program 3.4 shows how you can choose another folder, in this case the folder March2012, to store your work.

Program 3.4: Saving Output to a Custom Folder

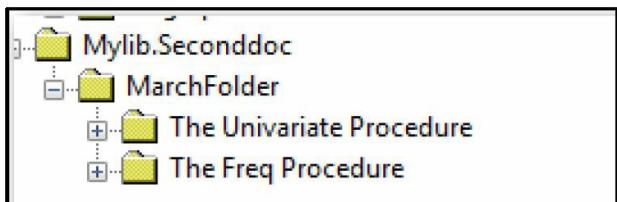
```
ods trace on;
ods document name=mylib.seconddoc dir=(path=\march2012);
proc univariate data=class;
by sex;
var height age;
run;

proc freq data=class;
table sex;
run;
ods document close;

proc document name=mylib.seconddoc;
list / levels=all;
run;
quit;
```

The output from this program shows the layout of folders. It is the same as shown in Display 3.1, except that now the folders are stored in the folder `March2012` instead of at the top-level folder. Display 3.2 shows the layout. The contents of the folders storing output from the UNIVARIATE procedure are the same as in Display 3.1, with one exception: these same folders are now stored as subfolders of the `March2012` folder. One application of this technique is that if you have monthly reports to produce, the DIR= option makes it easy to keep related SAS output organized within a single document.

Display 3.2: Storing SAS Output to a Current Folder



One application of the DIR= option is to permit you to change directories repeatedly without having to close and reopen the DOCUMENT destination. Program 3.5 shows the outline of a mini-analysis of a data set, and creates three directories: snapshot tables, graphs, and Analysis. You invoke the DOCUMENT statement with the DIR= option to change the current folder so that each output goes into the folder that you feel is most suitable.

Program 3.5: Creating Multiple Directories Using the ODS DOCUMENT Statement

```
ods document name=sample_doc dir=(path=\snapshot_tables label="Get Counts by categories");
proc freq data=work.cars;
```

```
run;  
/* same document open, but change directory only */  
ods document dir=(path=\graphs);  
  
proc sgplot data=work.cars;  
by origin type;  
scatter x=horsepower y=msrp;  
run;  
  
ods document dir=(path=\analysis label="Exploratory Analysis - Initial");  
proc univariate data=work.cars;  
run;  
ods document close;  
  
proc document name=sample_doc;  
list / levels=1;  
run;  
quit;
```

Behavior of the Pathname

In Chapter 2, “The ODS DOCUMENT Destination,” you learned how to read pathnames and object names from the ODS TRACE statement. You saw that there is a direct correspondence between ODS TRACE pathnames and ODS document paths.

When you are using the DIR= option, the only modification needed is that the output saved under that path is now done relative to the DIR specified in the DIR option. An example will make this clear.

Program 3.6: Determining Storage Locations When DIR= Is Specified

```
ods trace on;
ods document name=mylib.firstdoc
  dir=(path=\special ①
        label='All analysis pertaining to SASHELP.AIR');
  title 'Air Pollution Summary Report';
  proc means data=sashelp.air;
  run;
ods document close;
```

- ① Instructs the DOCUMENT destination to use the folder named `special` to store all output that was sent to the destination by the MEANS procedure.

The ODS TRACE statement shows that this item was produced:

```
Output Added:
-----
Name:      Summary
Label:     Summary statistics
Template:  base.summary
Path:      Means.Summary
-----
```

Using the method from Chapter 2, the ODS Path `Means.Summary` would be stored in the document as `means#\summary#1`. Since the base folder is now `special#1` because of the DIR= option used, the DOCUMENT destination will store output from the MEANS procedure at the following location:

`\Special#\Means#\Summary#1`

Without the DIR= option, it would be stored at the following path:

`\Means#\Summary#1`

The LABEL= Option

If you use the DIR= option to send SAS output to a specific folder, the LABEL= option enables you to provide a label for that folder. Program 3.7 performs the same tasks as Program 3.4. In this version, you add a more descriptive label.

Program 3.7: Adding a Descriptive Label to a Folder

```
ods document name=mylib.anotherdoc\dir=(path=\March2012 label='March 2012 Reporting');
/* CODE */
ods document close;
```

The ODS DOCUMENT statement can also apply a label to the entire document. Within the DIR= option, leave the PATH= option blank, but still specify a label with the LABEL= option, as shown in Program 3.8.

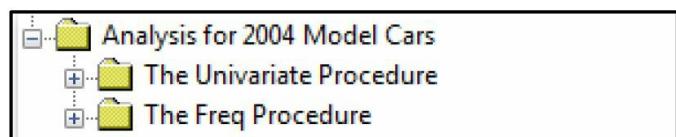
Program 3.8: Create a Document with a Custom Label

```
ods trace on;
```

```
ods html file="program3_8.html";
ods document name=mylib.lastone(write)
  dir=(label='Analysis for 2004 Model Cars');
proc univariate data=cars;
by type;
var weight msrp;
run;
proc freq data=cars;
table origin * type / list;
run;
ods document close;
proc document name=mylib.lastone;
list / levels=all;
run;
quit;
ods html close;
```

As with the other examples, the view of the document with labels is shown in Display 3.3.

Display 3.3: Document with a Custom Document Label



The CATALOG= Option

Because the CATALOG option requires more in-depth knowledge of SAS/GRAFH software, you can read more about it in the appendix. The basic idea is that if you want to replay SAS/GRAFH output, in certain circumstances, you might need to manage a graphics catalog. The REPLAY statement will then use that catalog to replay the graphs. However, with ODS Statistical Graphics, you can treat graphical output just like any other type of SAS output without having to learn special commands to handle graphics.

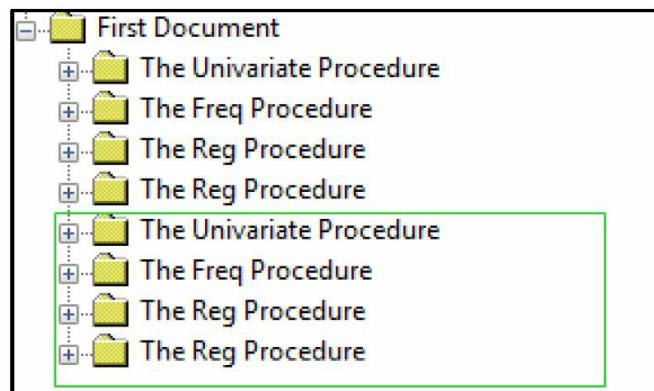
Using Access Options (Write and Update)

So far you have been using the default mode for adding output to the document. As you recall from Chapter 2, the document accumulates output and continues to do so even when closed and opened multiple times. This type of access has a name; it is called Update mode. Because Update mode is the default, you don't need to specify it, or be aware that you are using it.

Write access mode makes the document appear as if you were writing a new document each time that you open the DOCUMENT destination. ODS uses the term *write* slightly differently from how it is used in the computing world in general. You might want to think as if it were named *overwrite*. Opening a document with the Write access option will clear the entire ODS document of all of its data. If applied to a folder instead of the entire document, it will clear only that folder. Some examples will make this clear. Programs 3.9 through 3.11 provide these examples.

One way to see the difference is to resubmit Program 3.1. Since Program 3.1 opens the destination in Update access mode, the resulting document now contains two of each type of output. Display 3.4 illustrates the document's content when you use Update mode to save SAS procedure output to the document. Using Update mode is the default behavior, and it is what you want to use most of the time, since it does not destroy work.

Display 3.4: Behavior of Update Mode



If you wanted to rebuild the document from scratch, use Write access mode to clear the `mylib.firstdoc` document. Program 3.9 shows the DOCUMENT statement for Program 3.1 using Write access mode. The remainder of the program is the same as in Program 3.1.

Program 3.9: Using Write Access Mode

```
ods document name=mylib.firstdoc(write);
/* rest of code is identical to program 3.1 */
```

The ODS DOCUMENT statement instructs the system to erase the entire document and begin inserting SAS output to an empty document. This is one of the helpful uses of Write access mode. When you want to ensure that the document contains *only* the SAS output from a particular block of code, Write access mode is a good choice. But beware: there is absolutely no warning that you are emptying your document.

Program 3.10 is another example of this idea. If you run this code, you do not need to be concerned with what is already in this document. The results will always be the same. In this case, this is a single table from the TABULATE procedure.

Program 3.10: Deleting and Starting Over with a Clean Document

```
ods document name=mylib.onlytab(write);
/* some output */
proc tabulate data=sashelp.cars;
class origin;
table origin,n*f=5.; /* width=5, no dec. places */
run;
ods document close;
```

Write access mode can also be used to clear a single folder for rewriting. Program 3.11 shows how to overwrite one month of reporting data with an updated report.

Program 3.11: Clearing All SAS Output from a Subfolder

```
ods document name=reports.monthly
dir=(path=\march2012(write) label='New March Reports');
/* new report code goes here */
ods document close;
```

In most cases, it is better to use Update mode. Update mode is consistent with most people's expectations that the document should archive output, preserving as it goes. A less destructive version of Program 3.11 is shown next in Program 3.12.

Program 3.12: Saving New Work Along with the Old

```
ods document name=reports.monthly
dir=(path=\march2012\new_reports label='New March Reports');
/* code to create modified report */
ods document close;
```

Update mode is intended for most day-to-day practical work, because it does not destroy work. If you do not want to see certain output ever again, there are various ways to exclude that output from document replay. These are covered in Chapter 5, "The REPLAY Statement."

Example: Reviewing Your Progress

To check that the document was assembled correctly, you could check your work by closing your SAS session, reopening it, and executing the following DOCUMENT procedure code. However, there is really no need to do that unless you want to convince yourself that the document did in fact persist. Program 3.13 illustrates again that once the output is stored, you can replay it to any ODS destination without having to rerun the original analysis.

Program 3.13: Checking on the Document

```
ods listing; /* opens the once proud listing destination */
proc document name=mylib.firstdoc;
replay;
run;
quit;
```

You should see the same output from Program 3.9 as you did for Program 3.3, although you chose a different appearance by sending output from the DOCUMENT destination to the LISTING destination. Output 3.2 shows an excerpt of the output. In order to save space, some page breaks were suppressed. You will learn how to do that for yourself in Chapter 7, “Customizing Output.”

Output 3.2: A Version of the Stored Output Sent to the LISTING Destination

| The UNIVARIATE Procedure | | | |
|----------------------------|-------------|---------------------|------------|
| Variable: Height | | | |
| ----- SEX=F ----- | | | |
| Moments | | | |
| N | 9 | Sum Weights | 9 |
| Mean | 60.5888889 | Sum Observations | 545.3 |
| Std Deviation | 5.01832752 | Variance | 25.1836111 |
| Skewness | -0.7238643 | Kurtosis | -0.3464949 |
| Uncorrected SS | 33240.59 | Corrected SS | 201.468889 |
| Coeff Variation | 8.28258714 | Std Error Mean | 1.67277584 |
| Basic Statistical Measures | | | |
| Location | | Variability | |
| Mean | 60.58889 | Std Deviation | 5.01833 |
| Median | 62.50000 | Variance | 25.18361 |
| Mode | . | Range | 15.20000 |
| | | Interquartile Range | 7.80000 |
| Tests for Location: Mu0=0 | | | |
| Test | -Statistic- | -----p Value----- | |

The UNIVARIATE Procedure
Variable: Height

----- SEX=F -----

Moments

| | | | |
|-----------------|------------|------------------|------------|
| N | 9 | Sum Weights | 9 |
| Mean | 60.5888889 | Sum Observations | 545.3 |
| Std Deviation | 5.01832752 | Variance | 25.1836111 |
| Skewness | -0.7238643 | Kurtosis | -0.3464949 |
| Uncorrected SS | 33240.59 | Corrected SS | 201.468889 |
| Coeff Variation | 8.28258714 | Std Error Mean | 1.67277584 |

Basic Statistical Measures

| | Location | Variability | |
|--------|----------|---------------------|----------|
| Mean | 60.58889 | Std Deviation | 5.01833 |
| Median | 62.50000 | Variance | 25.18361 |
| Mode | . | Range | 15.20000 |
| | | Interquartile Range | 7.80000 |

Tests for Location: Mu0=0

Test -Statistic-----p Value-----

Summary

The ODS DOCUMENT statement opens a DOCUMENT destination to collect SAS output. You can save SAS output to an existing document using one of two access modes: Update mode, the default, adds output to an existing document, whereas Write mode clears a document and adds the output to an empty document. You can also open a particular path within a document using these two modes.

An ODS document can be envisioned as a sequence of folders, one for each SAS procedure. Folders can have subfolders, and output is organized hierarchically based on the ODS pathname. You can also create your own folders using the DIR= option.

In addition, you can run a procedure multiple times, and store that output in the same document. The document accomplishes this by means of a *sequence numbering* system that ensures that each path will be unique. Finally, close your document using the CLOSE statement.

Chapter 4: Listing Documents Using the DOCUMENT Procedure

[Introduction](#)

[The DOCUMENT Procedure](#)

[Terminology](#)

[Directory Notation](#)

[Root Level Directory](#)

[Current Directory](#)

[Invoking the DOCUMENT Procedure](#)

[The NAME= Option](#)

[The LABEL= Option](#)

[Running the Document Procedure Interactively](#)

[Access Modes](#)

[The LIST Statement](#)

[Simple Invocation of the LIST Statement](#)

[Sequence Numbers](#)

[LEVELS= Option](#)

[Using the DETAILS= Option](#)

[ORDER= Option](#)

[BYGROUPS Option](#)

[Warning: BYGROUPS Option Might Show No Output](#)

[Using WHERE= Clauses to Conditionally View a Document](#)

[WHERE Variables](#)

[Examples](#)

[Interlude](#)

[Debugging WHERE= Clauses](#)

[The WHERE= Clause and ODS Names](#)

[Comparison with the ODS SELECT and EXCLUDE Statements](#)

[_LABELPATH_ - Proceed at Your Own Risk DETAILS option is not](#)

[The SETLABEL Statement](#)

[Important: Labels Are Not Titles](#)

[Summary](#)

Introduction

This chapter explains how to review the contents of the ODS document and make the listings easy to read. The document can function as a journal of your project as it evolves. Therefore, making the LIST itself readable is an important prerequisite to replaying.

In order to view the output that is sent to an ODS document, you need to use the DOCUMENT procedure. The previous chapter covered how to create a document and store SAS procedure output. In the first chapter, you learned a rudimentary version of the LIST statement, provided to help you verify that your procedure output was stored as you intended.

This chapter takes the discussion further, discussing how to start the DOCUMENT procedure and providing a more complete discussion of the options that are available in the LIST statement.

Beginning with SAS 9.2, you can restrict which documents appear in the listing by using WHERE= clauses. This is analogous to how the WHERE= data set option lets you subset rows from data sets.

The DOCUMENT Procedure

The LIST statement provides your first detailed look at the DOCUMENT procedure, although it has been prevalent throughout the examples. The structure of the document has also been covered in previous chapters. This section covers the terminology and notation that are needed for understanding and effectively using the LIST and SETLABEL statements.

You can also use a graphical user interface to interact with your document. Accessing the DOCUMENT procedure with code is the emphasis in this book. However, you might want to review the GUI. You can find an excellent discussion in Andrew Karp's paper, cited in the references.

Terminology

The terms *folder* and *directory* are used interchangeably, depending on which is more natural in the context of discussion.

Directory Notation

The following notation is used in the book:

^ (one caret) The current directory. This directory will remain the root directory for this chapter and the next until the statements to change directories are discussed.

^^ (two carets) The Parent directory of the current directory. This directory is not needed in this chapter because the current directory will always be the root directory.

Root Level Directory

The top level of the document tree is called the *root*, and it is analogous to the root directory of a file system used by Windows or UNIX. For example, the directory denoted C:\ on most Windows systems and the backslash (/) on a UNIX system are also called root, or top-level, directories. When the DOCUMENT procedure starts, the default directory is this root-level directory. The root directory can be referred to with the character strings \ or \libname.memname\. It is not necessary to list the initial backslash if you want to list a subdirectory of the root directory. So, you could submit any of the following to obtain a list of the top-level directory.

```
list ;
list \;
list ^; /* if no change of directory has been made */
list \mylib.firstdoc;
```

Of these, the first form is the briefest. The second and third are useful when it is required to provide a pathname, such as with WHERE= clauses. WHERE= clauses are discussed later in the chapter. The second form specifies the top-level directory, and the third specifies the current directory. This is the root directory, provided you have not changed the directory. The second and third forms are the same when the DOCUMENT procedure starts. The fourth expression of the root directory is called an *absolute pathname*. This form is useful when you are copying or moving documents. Examples of the first three will be used in this chapter, and absolute pathnames and their associated examples are covered in Chapter 6, “Managing Folders.”

Current Directory

When the DOCUMENT procedure starts, the current directory is set to the top-level, or root, directory. This can be changed, but for the next few chapters, the current directory will remain the top-level directory until more document management statements are covered in Chapter 6.

Invoking the DOCUMENT Procedure

The first goal of this chapter is to show how to start the DOCUMENT procedure and obtain a listing of all of the output that is stored within that document.

The syntax of the DOCUMENT procedure statement is as follows:

```
PROC DOCUMENT options;
```

The PROC DOCUMENT statement has the following options:

```
NAME=memname.libname(access option)
```

```
LABEL=quoted text
```

The DOCUMENT procedure, like its functional cousin the ODS DOCUMENT statement, also has a NAME= and LABEL= option, and their function is similar in the DOCUMENT procedure.

The NAME= Option

The NAME= option specifies the current document for the statements that are issued in the DOCUMENT procedure. All DOCUMENT procedure statements require a *current document*. The sole exception is the DOC statement. If you do not provide one, most DOCUMENT procedure statements will return an error, as Display 4.1 demonstrates. Note that in the code submitted, there was no current document when the LIST statement was submitted.

Display 4.1: The LIST Statement without a Current Document

```
1 proc document;
2   LIST;
3   RUN;

ERROR: You have not specified a current document.
4 QUIT;

NOTE: PROCEDURE DOCUMENT used (Total process time):
      real time          0.00 seconds
      cpu time          0.00 seconds
```

You can specify a current document either by using the NAME= option or by providing an absolute pathname, as in the following program:

```
proc document;
/* mylib first doc temporarily becomes the current doc */
list 'mylib.firstdoc';
run;
```

It is easier to provide a current document so that you do not have to enter the absolute pathname for every command.

Unlike some SAS procedures that take the last DATA set used as a default argument, there is no such default for the current document in the DOCUMENT procedure. Therefore, remember to provide a current document.

Here is the syntax for the NAME= option:

```
NAME=<libref.>member-name(access option)
```

Program 4.1 shows a simple but useful application of the DOCUMENT procedure: namely, the listing and replaying of a document. This provides a table of contents along with the replayed output.

Program 4.1: Starting the DOCUMENT Procedure Using the NAME= Option

```
proc document name='mylib.firstdoc';
/* update is the default access mode */
list / details; /* details option needed to see labels */;
run;
replay;
run;
quit;
```

The listing part of the output from this program is shown in Output 4.1. In order to fit the table on the page, some columns were omitted. Your output may have more columns.

Listing of: \Mylib.Firstdoc\

Order by: Insertion

Number of levels: 1

| Obs | Path | Type | Size in Bytes | Created | Modified | Symbolic Link | Template | Label | Page Break |
|-----|---------------|------|---------------------|--------------------|--------------------|------------------|----------|--|---------------|
| 1 | \Univariate#1 | Dir | | 14AUG2012:05:45:01 | 14AUG2012:05:45:02 | | | The Univariate Procedure | |
| 2 | \Freq#1 | Dir | | 14AUG2012:05:45:02 | 14AUG2012:05:45:02 | | | The Freq Procedure | |
| 3 | \special#1 | Dir | | 14AUG2012:05:45:02 | 14AUG2012:05:45:02 | | | All analysis pertaining to SASHELP.AIR | |

The NAME= option provides the name of the current document that you want to manage. If you enter the name of a document that does not yet exist, an empty document will be created with that name, and this empty document will become current.

The LABEL= Option

The LABEL= option is one of several ways to add a descriptive label to your document. The other two that are covered in this chapter use the SETLABEL statement in the root directory and the LABEL= option in the DOC statement. You will see examples of the other two methods later in this chapter. Here is the syntax for the LABEL= statement option:

```
LABEL="label-text" | 'label-text'
```

Returning to the matter of labeling a document, Program 4.2 shows how you can relabel an existing document.

Program 4.2: Relabeling an Existing Document

```
proc document name=my.lib.firstdoc label="Let's call this the  
SASHELP.CLASS analysis, part I.";  
doc lib=my.lib;  
run;  
quit;
```

This DOC statement produces a list of all documents in the library specified by the LIB= option. The full discussion of the DOC statement may be found in “Browsing Collections of Documents with the DOC Statement,” in Chapter 6, “Managing Folders.” Output 4.2 shows the existing documents in the library, along with the newly applied label for MYLIB.FIRSTDOC.

Output 4.2: Verifying that the Label was Successfully Applied

List of Documents with Labels Applied

| Documents | | |
|-----------|------------------|---|
| Obs | Name | Label |
| 1 | Mylib.Firstdoc | Let's call this the SASHELP.CLASS analysis, part I. |
| 2 | Mylib.Quickstart | |

Running the Document Procedure Interactively

The DOCUMENT procedure runs interactively. This means that the DOCUMENT procedure assumes you will be submitting several statements when you execute the procedure. The most important implication for your code is that you must submit a RUN statement in order for the DOCUMENT procedure to accept your statements. It will then process all statements that were issued since the last RUN statement or since the procedure was started. The code that exists between two RUN statements (or the DOCUMENT procedure statement and a RUN statement) is called a *RUN group*. This definition holds for other procedures as well. It is not limited to the DOCUMENT procedure.

In this chapter, each example is written so that it can be run as a self-contained unit. However, it is not necessary to submit PROC DOCUMENT and QUIT statements as often as directed by the programs in this chapter. An example of how the DOCUMENT procedure looks when you submit statements in an interactive environment is presented in the appendix. Whether you run code interactively, or you run code by submitting it to a remote server, you should be aware of RUN groups. While the DOCUMENT procedure is running, any code that you submit will be considered as DOCUMENT procedure code until the QUIT statement is encountered. The RUN groups will be pointed out as they appear in the examples. If you execute the RUN statement after each command, you can be sure that all your statements are executed.

Access Modes

The PROC DOCUMENT statement has the two access modes that you learned about in the previous chapter, as well as Read mode.

Read mode is probably the safest for listing and replaying existing work. Although you are restricted because you cannot make any changes to titles or document properties, you also have the security of knowing that you cannot alter your document.

Write access mode is for creating new documents, or for wiping documents clean and starting over. One way to remember this mode is to think of it as *overwrite* mode. Write mode is useful for situations where you might want to start with a clean slate.

Update access mode is the default access mode. It permits you to make changes to the structure of your document, copy folders, and add notes and annotations as well as set labels of subdirectories.

The LIST Statement

Here is the syntax for the LIST command:

```
LIST path1,path2,...pathn / options;
```

The following options are covered in this chapter:

```
LEVELS=n|ALL  
BYGROUPS  
DETAILS  
ORDER=DATE|INSERT|ALPHA
```

There is also a FOLLOW option, which is covered in Chapter 10, “Using Links.”

Simple Invocation of the LIST Statement

The simplest use of the LIST statement option that is possible in a document is illustrated in Program 4.3. This program uses the document that was created in Programs 3.1 and 3.4 in Chapter 3, “The ODS DOCUMENT Destination.”

Program 4.3: Top-Level List of a Document

```
title 'Simple Listing of MYLIB.FIRSTDOC';
proc document name=mylib.firstdoc;
  list;
  run;
quit;
```

The output is a listing of all the folders, one for each procedure, as shown in Output 4.3.

Output 4.3: Output from a Simple LIST Statement

| Simple Listing of MYLIB.FIRSTDOC | | |
|----------------------------------|--------------|------|
| Listing of: \Mylib.Firstdoc\ | | |
| Order by: Insertion | | |
| Number of levels: 1 | | |
| Obs | Path | Type |
| 1 | Univariate#1 | Dir |
| 2 | Freq#1 | Dir |
| 3 | special#1 | Dir |

Returning to the output of the LIST statement, notice that each SAS procedure is stored in its own directory and that there are two directories for output from the REG procedure. The reason for that is that the original programs from Chapter 3, “The ODS DOCUMENT statement,” executed the REG procedure twice. Each procedure, in addition to the pathname that is displayed in the directory, has a *sequence number* after it in order to make it unique. Notice also that the name of the directory being listed is at the top of the output.

Sequence Numbers

Sequence numbers were briefly referenced in the previous chapter, and require a more detailed discussion now. The ODS document attaches a sequence number to every pathname component in order to ensure that pathnames remain unique. This permits an ODS document to store several folders with the same name, as illustrated by Program 4.4.

Program 4.4: Illustrating Sequence Numbers

```
ods document name=mylib.firstdoc;
/* re-run program 3.1 again if necessary */
proc reg data=class;
by sex;
model weight = age height;
run;
quit;

proc reg data=class;
model weight = age height sex;
run;
quit;
ods document close;
```

In this program, the first `reg` entry, `reg#1`, is assigned a sequence number of #1. The second folder, holding output from the REG procedure, `reg#2`, is assigned a sequence number of #2. Sequence numbers are increased each time you run a procedure with the same name as an existing file in the folder where ODS will store your procedure output.

Sequence numbers are *never* reused unless *all* entries with the same name are deleted. In that case, the sequence counter reverts to 1. Since you will not be deleting anything in this chapter, the appropriate examples of this phenomenon are deferred to Chapter 6, “Managing Folders.”

Program 4.5 creates another example document with several copies of output from the REG procedure. It is not necessary to understand what this code from these procedures is designed to do; it suffices to understand that this might represent a common scenario.

Program 4.5: Several Folders Might Have the Same Name

```
ods document name=mylib.morecars;
ods html file="program4_5.html";
libname mylib "c:\users\Michael\DocumentBook";
proc sort data=sashelp.cars out=WORK.cars;
/* nothing against Hybrids, I promise! */
where upcase(type) ne 'HYBRID';
by origin type;
run;

title1 "Explore inverse relationship between weight and mileage";
data cars;
set cars;
invw = 1/weight;
run;
proc reg data=cars ;
model mpg_highway = invw;
run;
quit;
title "Are people paying more for high mileage?";
proc reg data=cars;
model msrp = mpg_highway;
run;
quit;
title "Maybe it's something else";
proc reg data=cars;
model msrp = mpg_highway weight horsepower cylinders;
run;
quit;
```

```

title "Same model as #3 with more diagnostic plots";
proc reg data=cars;
model msrp = mpg_highway weight horsepower cylinders / all;
run;
quit;
ods document close;
proc document name=mylib.morecars;
list;
run;
quit;
ods html close;

```

Output 4.4: Listing Output of TEMP Document

Four Regression Analyses of Cars Dataset

| Listing of: \Work.Temp\ | | |
|-------------------------|--------------|------|
| Order by: Insertion | | |
| Number of levels: 1 | | |
| Obs | Path | Type |
| 1 | Reg#1 | Dir |
| 2 | Reg#2 | Dir |
| 3 | Reg#3 | Dir |
| 4 | Reg#4 | Dir |

If you run any statement from the DOCUMENT procedure that takes a path argument, and you do not specify a sequence number, the DOCUMENT procedure defaults to the current sequence number, which for this example is #4. Program 4.6 illustrates omitting the sequence number.

Program 4.6: Replying without Sequence Number

```

ods html file="program4_6.html";
proc document name=mylib.quickstart;
list reg;
run;
replay reg;
run;
quit;
ods html close;

```

Program 4.6 produces output from the folder `reg#4`, which is the folder with the current, and also the highest, sequence number.

Output 4.5: Effect of Omitting the Sequence Number

Same model as #3 with more diagnostic plots

The REG Procedure
Model: MODEL1

| Model Crossproducts X'X X'Y Y'Y | | | | | | | |
|---------------------------------|---------------|-----------|-------------|------------|------------|-----------|-------------|
| Variable | Label | Intercept | MPG_Highway | Weight | Horsepower | Cylinders | MSRP |
| Intercept | Intercept | 423 | 11272 | 1517810 | 91688 | 2463 | 13914978 |
| MPG_Highway | MPG (Highway) | 11272 | 311720 | 39077514 | 2340526 | 63256 | 350978472 |
| Weight | Weight (LBS) | 1517810 | 39077514 | 5687390466 | 343255518 | 9203602 | 52703717623 |

From a practical point of view, this means that if you just finished running code, a path entirely without sequence numbers will give you the most recent output.

If you want to list or replay output from a folder that has a sequence number lower than the current sequence number, you must include the sequence number. Program 4.7 demonstrates this principle by listing and replaying the contents of the `reg#1` folder.

Program 4.7: Using a Sequence Number to Replay an Older Version of the Document

```
proc document name=mylib.firstdoc;
  list reg#1;
  run;
quit;
ods html close;
```

LEVELS= Option

The LIST statements shown so far produced only the objects that are at the top level of the directory that was requested. You have seen LEVELS=ALL in the previous chapter. More generally, if you want to see a specific number of levels of subfolders, use the LEVELS= option. The default is LEVELS=1, which displays only a top-level table of contents, as you saw in Output 4.4.

The number of levels deep is counted based on the level of the path that was provided to the LIST statement. Program 4.8 shows all of the subfolders at the first level, and then all levels beneath Univariate#1.

Another purpose of Program 4.8 is to create some PROC UNIVARIATE output that includes graphical output. This example will be used again for Program 4.18.

Program 4.8: Levels=1 Starting with a Folder Name

```
ods html file="program4_8.sas";
/* data=cars was defined in program 1.1 */
ods document name=temp2 ;
proc univariate data=cars normaltest;
by origin type;
var msrp weight horsepower;
cdfplot msrp;
run;
ods document close;
ods listing;
/* here's the real action */
proc document name=temp2;
list univariate / levels=1;
run;

list
ods html close;
```

The output from Program 4.8 (see Output 4.6) is not terribly interesting to view, but it merits some discussion precisely because it is not as informative or as helpful as it could be. If the document is to be useful in helping you retrieve output, you have to know which output is which. A numbered list of BY-groups is not reader-friendly. There are several options to provide the extra detail needed, which are described in the following sections.

Quick Check-in Question

What is the highest sequence number shown in Output 4.6? The answer to the quick quiz is at the back of the chapter, after the summary.

Output 4.6: A Situation Where the LEVELS=1 Output Is Not So Informative

Univariate Output

| Listing of: \Mylib.Quickstart\Univariate#1 | | |
|--|------------|------|
| Order by: Insertion | | |
| Number of levels: 1 | | |
| Obs | Path | Type |
| 1 | ByGroup1#1 | Dir |
| 2 | ByGroup2#1 | Dir |
| 3 | ByGroup3#1 | Dir |
| 4 | ByGroup4#1 | Dir |
| 5 | ByGroup5#1 | Dir |
| 6 | ByGroup6#1 | Dir |

In contrast to the quick overview provided by the LEVELS=1 option, the other extreme is to show every path in the document. The LEVELS=ALL option is the option that you want in this case. LEVELS=ALL output can be quite voluminous, as you might recall from the previous chapter's examples.

Output 4.7: Listing the Entire Directory Tree Below a Specific Path

| Listing of: \Work.Temp\Univariate#1 | | |
|-------------------------------------|---|-------|
| Order by: Insertion | | |
| Number of levels: All | | |
| Obs | Path | Type |
| 1 | \Univariate#1\ByGroup1#1 | Dir |
| 2 | \Univariate#1\ByGroup1#1\MSRP#1 | Dir |
| 3 | \Univariate#1\ByGroup1#1\MSRP#1\Moments#1 | Table |
| 4 | \Univariate#1\ByGroup1#1\MSRP#1\BasicMeasures#1 | Table |
| 5 | \Univariate#1\ByGroup1#1\MSRP#1\TestsForLocation#1 | Table |
| 6 | \Univariate#1\ByGroup1#1\MSRP#1\TestsForNormality#1 | Table |
| 7 | \Univariate#1\ByGroup1#1\MSRP#1\Quantiles#1 | Table |
| 8 | \Univariate#1\ByGroup1#1\MSRP#1\ExtremeObs#1 | Table |
| 9 | \Univariate#1\ByGroup1#1\MSRP#1\CDFPlot#1 | Dir |
| 10 | \Univariate#1\ByGroup1#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph |

Output 4.7 shows the listing for the `Univariate#1` folder. This listing is generated by submitting Program 4.9. The omission of a pathname implies that you want to list all subfolders of the current directory. If you have not changed the current directory (and no means has yet been provided to do that), then the current directory is the root directory and omitting the pathname therefore results in listing the entire document.

Program 4.9: All Levels of the Entire Document

```
proc document name=temp(read);
list univariate#1 / levels=all;
run;
quit;
```

The most complete listing, showing all paths of the document as well as properties and labels for every object and folder, combines LEVELS=ALL with the DETAILS option. That is covered next.

The output from Program 4.9 is not any more instructive than that of Output 4.8, and is therefore omitted.

You can include more than one pathname on the list command, thereby listing only the items that you feel are of greatest interest. Other times, you might want to show output that was hidden at the bottom. Program 4.10 shows an example using the `mylib.firstdoc` document.

Program 4.10: Listing More Than One Folder

```
ods html file="program4_10.html";
proc document name=mylib.firstdoc;
  title "When you list pathnames separated by a comma, separate
tables";
  title2 "Are generated";
  list freq,reg#1 / levels=3;
run;
quit;
ods html close;
```

Notice that Output 4.8, the output from Program 4.10, has folders that are four deep. How can this be if you used LEVELS=3? Notice that you are starting with \FREQ#1, and there are three levels beneath that, which accounts for why you see folders in the list that have four components. LEVELS=3 was a good choice, striking a balance between detail and size. As you get to know the SAS procedures that you use most frequently, you will get a sense of which LEVELS=option works best for listing your output.

Output 4.8: Listing More Than One Folder

***When you list pathnames separated by a comma, separate tables
Are generated***

| Listing of: \Mylib.Firstdoc\Freq#1 | | |
|------------------------------------|--------------------------------|-------|
| Order by: Insertion | | |
| Number of levels: 3 | | |
| Obs | Path | Type |
| 1 | \Freq#1\Table1#1 | Dir |
| 2 | \Freq#1\Table1#1\OneWayFreqs#1 | Table |

Listing of: \Mylib.Firstdoc\Reg#1

Order by: Insertion

Number of levels: 3

| Obs | Path | Type |
|-----|---|------|
| 1 | \Reg#1\ByGroup1#1 | Dir |
| 2 | \Reg#1\ByGroup1#1\MODEL1#1 | Dir |
| 3 | \Reg#1\ByGroup1#1\MODEL1#1\Fit#1 | Dir |
| 4 | \Reg#1\ByGroup1#1\MODEL1#1\ObswiseStats#1 | Dir |
| 5 | \Reg#1\ByGroup2#1 | Dir |
| 6 | \Reg#1\ByGroup2#1\MODEL1#1 | Dir |
| 7 | \Reg#1\ByGroup2#1\MODEL1#1\Fit#1 | Dir |
| 8 | \Reg#1\ByGroup2#1\MODEL1#1\ObswiseStats#1 | Dir |

Using the DETAILS= Option

You can see additional properties of your stored SAS output by using the DETAILS option in the LIST statement. Some of these details have not yet been discussed. For those, the appropriate chapter is also given. With the exception of the OBS column, the meaning of which should be self-evident, the descriptions of the columns in the detailed listing are as follows. Some columns are not relevant at this point since the concepts behind them will be discussed in later chapters.

Recall the listing from Output 4.4. It was not helpful because it contained no reader-friendly descriptions of what the folder contains. The DETAILS option can remedy this because the default labels provided by the procedure might be helpful in recalling the folder's contents. Program 4.11 demonstrates the same LIST statement as that in Program 4.6, using the DETAILS option.

Program 4.11: A Detail Listing of the Univariate Tree, at the Highest Folder Level

```
ods html file="baobab.html";
title "Simple Detailed More Listing";
proc document name=mylib.morecars(update);
list univariate#1 / levels=1 details;
run;
quit;
ods html close;
```

Output 4.9 shows the listing. Notice the columns that were not in the simpler listing.

Output 4.9: A Detailed LISTING

| Detailed Listing | | | | | | | | |
|--|------------------------------|------|---------------------|--------------------|--------------------|------------------|----------|----------------------------|
| Listing of: \Mylib.Morecars\Univariate#1 | | | | | | | | |
| Order by: Insertion | | | | | | | | |
| Number of levels: 1 | | | | | | | | |
| Obs | Path | Type | Size in Bytes | Created | Modified | Symbolic Link | Template | Label |
| 1 | \Univariate#1 \ByGroup1#1 | Dir | | 12AUG2012:21:16:41 | 12AUG2012:21:16:41 | | | Origin=Asia Type=SUV |
| 2 | \Univariate#1 \ByGroup2#1 | Dir | | 12AUG2012:21:16:41 | 12AUG2012:21:16:41 | | | Origin=Asia Type=Sedan |
| 3 | \Univariate#1 \ByGroup3#1 | Dir | | 12AUG2012:21:16:41 | 12AUG2012:21:16:41 | | | Origin=Asia Type=Sports |
| 4 | \Univariate#1 \ByGroup4#1 | Dir | | 12AUG2012:21:16:41 | 12AUG2012:21:16:41 | | | Origin=Asia Type=Truck |
| 5 | \Univariate#1 \ByGroup5#1 | Dir | | 12AUG2012:21:16:41 | 12AUG2012:21:16:41 | | | Origin=Asia Type=Wagon |

The meanings of the columns are outlined in Table 4.1. The meanings of OBS and SIZE in BYTES should be self-evident. In Table 4.1, Objects that are output objects (and therefore have their own titles and footnotes among other properties) are shaded.

Table 4.1: Meaning of Columns in the DETAIL List Output

| | | | | | | | | | | | | | | | | | |
|----------------------|---|-----|-----------|-------|--------------|-------|---------|------|--|--------|--|------|--|----------|------------------------------|----------|---|
| Path | The pathname of the object, relative to the current folder, which is listed at the top of the listing. | | | | | | | | | | | | | | | | |
| Type | <p>The type of the object. Some possible types include:</p> <table border="1"> <tr> <td>DIR</td><td>Directory</td></tr> <tr> <td>TABLE</td><td>Output table</td></tr> <tr> <td>GRAPH</td><td>A graph</td></tr> <tr> <td>LINK</td><td>A symbolic link (Covered in Chapter 10, “Working with Links.”)</td></tr> <tr> <td>REPORT</td><td>Report table from the REPORT procedure</td></tr> <tr> <td>NOTE</td><td>A free floating text note. (Covered in Chapter 7, “Customizing Output.”)</td></tr> <tr> <td>EQUATION</td><td>A specialized output object.</td></tr> <tr> <td>CROSSTAB</td><td>The CROSSTAB type applies only to the output object generated by the PROC FREQ TABLE statement.</td></tr> </table> | DIR | Directory | TABLE | Output table | GRAPH | A graph | LINK | A symbolic link (Covered in Chapter 10, “Working with Links.”) | REPORT | Report table from the REPORT procedure | NOTE | A free floating text note. (Covered in Chapter 7, “Customizing Output.”) | EQUATION | A specialized output object. | CROSSTAB | The CROSSTAB type applies only to the output object generated by the PROC FREQ TABLE statement. |
| DIR | Directory | | | | | | | | | | | | | | | | |
| TABLE | Output table | | | | | | | | | | | | | | | | |
| GRAPH | A graph | | | | | | | | | | | | | | | | |
| LINK | A symbolic link (Covered in Chapter 10, “Working with Links.”) | | | | | | | | | | | | | | | | |
| REPORT | Report table from the REPORT procedure | | | | | | | | | | | | | | | | |
| NOTE | A free floating text note. (Covered in Chapter 7, “Customizing Output.”) | | | | | | | | | | | | | | | | |
| EQUATION | A specialized output object. | | | | | | | | | | | | | | | | |
| CROSSTAB | The CROSSTAB type applies only to the output object generated by the PROC FREQ TABLE statement. | | | | | | | | | | | | | | | | |
| Created and Modified | SAS Date time values for when this object was created and modified. | | | | | | | | | | | | | | | | |
| Symbolic Link | If a link, the pathname of the actual object pointed to by the link. Links are discussed in Chapter 10. For the examples in this chapter, this column will be blank. | | | | | | | | | | | | | | | | |
| Template | If the output object uses an ODS Template to control its appearance, it is mentioned here. Templates are discussed in Chapter 11, “Working with Templates.” You do not need to be concerned with templates in order to use the DOCUMENT procedure effectively. | | | | | | | | | | | | | | | | |

The DETAILS option is not restricted to situations when LEVELS=ALL. It can be used in combination with any level of listing. Output 4.7 shows each folder with its default label. Having this label accessible makes the document listing more readable. Some fields, such as template, are blank because they are applicable to output objects, not to folders.

ORDER= Option

The default order is to list the document in the order in which the entries were added to the document. This order can be specified by using ORDER=INSERT, or by omitting the ORDER option altogether. INSERT is the default.

ORDER=ALPHA displays the listing in alphabetical order by pathname. This option is particularly helpful when looking within the output folder for a specific procedure. For example, it is easier to see the fit\weight output if the paths are displayed alphabetically.

ORDER=DATE is displayed in ascending order by creation date.

For the examples that are outlined in the first five chapters, ORDER=DATE produces output that is equivalent to ORDER=INSERT because output is inserted in the order that it was created. If you change the default order thorough MOVE or COPY statements, these two orderings might be different.

Program 4.12 shows some of the options. There is no option that uses descending order. ORDER is always ascending.

As a side note, Program 4.12 uses a different title for each LIST statement output. This approach works because the DOCUMENT procedure is interactive, and can support multiple RUN groups. This also means that you can add a different title for each RUN group. This is not a feature of the DOCUMENT procedure in particular, but of RUN groups in general.

Program 4.12: Possibilities for the ORDER= Option

```
ods html file="program 4_12.html";
proc document name=temp;
  title "Alpha Order";
  list / levels=1 details order=alpha;
run;
title "Insert Order";
  list / levels=1 details order=insert;
run;
title "Date Order";
  list / levels=1 details order=date;
run;
quit;
ods html close;
```

BYGROUPS Option

You have learned how to create listings at various levels of detail, from a top-level list of folders at the lowest level of detail, down to a complete listing of every path and every detail of each path. Along the way, you saw how adding the DETAILS option can help you understand the contents of your document.

BYGROUPS is another option that makes reviewing output easier. As the name suggests, it is possible only when BY-group processing is used to create the SAS output that is stored in the document. When you add the BYGROUPS option to a LIST statement, the system adds an additional column for each BY-group variable.

Using the BYGROUPS option produces output only if it is listing a level of the directory tree that has output objects. For this reason, to ensure that all output is captured, it is safest to use this in conjunction with the LEVELS=ALL option.

Program 4.13: Entire Document Listing Including BY-Group Variables

```
ods html file="program4_13.html";
proc document name=my.lib.morecars(read); ①
  title "Univariate Output";
  list univariate / levels=all by groups; ②
run;
quit;
ods html close;
```

- ① Program 4.13 illustrates running the DOCUMENT procedure in Read mode. This ensures that nothing can be modified accidentally. This is a useful option while running the LIST and REPLAY operations.
- ② The BYGROUPS option directs the LIST statement to display output objects that carry BY-group information. Only output objects are shown when you use the BYGROUPS option. Recall that output objects can be of type TABLE, GRAPH, CROSSTAB, REPORT, or EQUATION. It is necessary only to identify output objects. Most of the examples in the book will cover TABLE and GRAPH. As far as the DOCUMENT procedure is concerned, one output object is as easy to handle as any other is. The code that produced the BY-groups was the first REG procedure in Program 3.4.

Output 4.10 shows the output objects along with their BY-group variables. This makes it very easy to identify and retrieve the relevant output.

Output 4.10: Showing BY-Group Variables for Output Objects

Univariate Output

Listing of: \Mylib.Morecars

Order by: Insertion

Number of levels: All

| Obs | Path | Type | Origin | Type |
|-----|---|-------|--------|------|
| 1 | \Univariate#1\ByGroup1#1\MSRP#1\Moments#1 | Table | Asia | SUV |
| 2 | \Univariate#1\ByGroup1#1\MSRP#1\BasicMeasures#1 | Table | Asia | SUV |
| 3 | \Univariate#1\ByGroup1#1\MSRP#1\TestsForLocation#1 | Table | Asia | SUV |
| 4 | \Univariate#1\ByGroup1#1\MSRP#1\TestsForNormality#1 | Table | Asia | SUV |
| 5 | \Univariate#1\ByGroup1#1\MSRP#1\Quantiles#1 | Table | Asia | SUV |
| 6 | \Univariate#1\ByGroup1#1\MSRP#1\ExtremeObs#1 | Table | Asia | SUV |
| 7 | \Univariate#1\ByGroup1#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | Asia | SUV |
| 8 | \Univariate#1\ByGroup1#1\Weight#1\Moments#1 | Table | Asia | SUV |
| 9 | \Univariate#1\ByGroup1#1\Weight#1\BasicMeasures#1 | Table | Asia | SUV |
| 10 | \Univariate#1\ByGroup1#1\Weight#1\TestsForLocation#1 | Table | Asia | SUV |
| 11 | \Univariate#1\ByGroup1#1\Weight#1\TestsForNormality#1 | Table | Asia | SUV |
| 12 | \Univariate#1\ByGroup1#1\Weight#1\Quantiles#1 | Table | Asia | SUV |

By contrast, Program 4.14 shows the same listing without the BYGROUPS option.

Program 4.14: Without the BYGROUPS Option

```
proc document name=my.lib.firstdoc;
  ods html file="example4_14.html";
  title "No By Groups in listing";
  list univariate / levels=all;
run;
```

This produces Output 4.11 (excerpted). Although the document still maintains the BY-group information, Output 4.9 does not provide columns to clarify. To make sure your listings show BY-group variables, use either or both of the BYGROUPS and DETAILS options. The difference between the two is that BYGROUPS only shows output objects, and puts each BY group in a separate column of the table. The DETAILS option shows the BY-group as a single text field, as in Output 4.7.

Output 4.11: Displaying BY-Grouped Output without the BYGROUPS Option

Listing of: \Mylib.Morecars\Univariate#1

Order by: Insertion

Number of levels: All

| Obs | Path | Type |
|-----|---|-------|
| 1 | \Univariate#1\ByGroup1#1 | Dir |
| 2 | \Univariate#1\ByGroup1#1\MSRP#1 | Dir |
| 3 | \Univariate#1\ByGroup1#1\MSRP#1\Moments#1 | Table |
| 4 | \Univariate#1\ByGroup1#1\MSRP#1\BasicMeasures#1 | Table |

Warning: BYGROUPS Option Might Show No Output



If there are no output objects carrying BY-groups among the paths selected, using the BYGROUPS option will produce *no output*. Therefore, if your document programs are producing no output, verify that you are using LEVELS=ALL and omit the BYGROUPS option. If reinserting the option causes the output to disappear, you've found the problem.

Program 4.15: No Output

```
/* write access mode ensures that you start with an empty document */
ods document name=explode(write);
proc univariate data=sashelp.retail;
run;
ods document close;
proc document name=explode;
list / levels=all details by groups;
run;
quit;
```

In this example, the DOCUMENT procedure code produces no output because there are no output objects that were created with BY-group variables.

If the document has some output objects with BY-groups and some without, the LIST statement will list only those with BY-groups, as Program 4.16 shows.

Program 4.16: Only Output with BY-Groups is shown

```
ods html file="program4_16.html";
proc document name=mylib.firstdoc;
list / by groups levels=all;
run;
quit;
ods html close;
```

An excerpt of the output is shown in Output 4.12. The LIST statement did not list any objects from the FREQ procedure, which did not use BY-group variables.

Output 4.12: Listing Output Objects That have a BY Group (Excerpt)

| No FREQ | 18 \Univariate#1\ByGroup2#1\Age#1\TestsForLocation#1 | Table M | |
|-------------------------------------|---|---------|-----|
| | 19 \Univariate#1\ByGroup2#1\Age#1\Quantiles#1 | Table M | |
| Procedure Output | 20 \Univariate#1\ByGroup2#1\Age#1\ExtremeObs#1 | Table M | |
| Listing of: \Mylib.Firstdoc\ | | | |
| Order by: Insertion | | | |
| Number of levels: All | | | |
| Obs | Path | Type | Sex |
| 1 | \Reg#1\ByGroup1#1\MODEL1#1\Fit#1\Weight#1\NObs#1 | Table | F |
| 2 | \Reg#1\ByGroup1#1\MODEL1#1\Fit#1\Weight#1\ANOVA#1 | Table | F |
| 3 | \Reg#1\ByGroup1#1\MODEL1#1\Fit#1\Weight#1\FitStatistics#1 | Table | F |

Using WHERE= Clauses to Conditionally View a Document

The WHERE statement for the DOCUMENT procedure, new to SAS 9.2, makes it easier to manage large document listings as well as replay documents. You can conditionally control which subsets of the directory tree will be listed. The WHERE statement uses syntax similar to the WHERE statement in the SAS DATA step. WHERE clauses can be applied to directories in LIST statements. They might also be used be applied in the REPLAY, COPY TO, DELETE, and MOVE TO statements.

WHERE Variables

The following special document variables can be used to conditionally select output when using the WHERE statement:

Table 4.2: Subsetting Variables

| | |
|--|---|
| <code>_CDATE_</code> | Creation Date |
| <code>_CDATETIME_</code> | Creation Date/Time |
| <code>_CTIME_</code> | Creation Time |
| <code>_LABELPATH_</code> | Pathname, in LABEL format |
| <code>_LABEL_</code> | The object label |
| <code>_MDATE_</code> , <code>_MDATETIME_</code> , <code>_MTIME_</code> | Similar to Cdate variables, but cover modification time. |
| <code>_NAME_</code> | Name of current entry. The name is the rightmost entry of the pathname. |
| <code>_PATH_</code> | Pathname of current entry |
| <code>_SEQNO_</code> | Sequence number of current entry |
| <code>_TYPE_</code> | Type of object: TABLE or DIRECTORY |
| <code>_MAX_</code> | See Chapter 5, “The REPLAY Statement.” |
| <code>_MIN_</code> | See Chapter 5, “The REPLAY Statement.” |
| <code>_OBS_</code> | Observation-number |
| Variable name | Variable name. Any variable name in the output object can be used. |

The `_max_`, `_min_`, `_obs_` and variable name keywords will be discussed in Chapter 5, “The REPLAY Statement.”

Examples

Program 4.17 shows how to find all analyses pertaining to the variable Height. In this case, only the `univariate#1` folder has paths that contain the variable Height. Notice that in all of these examples, it is necessary to insert an explicit pathname since the WHERE option requires a path. Note that the question mark (?) is the Contains operator. Also, the caret (^), between the list statement and the options, does not imply negation. In this context, it represents the *current directory*.

When you search starting at the current directory with a WHERE= clause, you must provide a path. You cannot leave the path blank to default to the current directory. For this reason, if you want to use the current directory as the base path for your WHERE= clause, you must include a marker for the current directory. This marker is the caret.

Program 4.17: Examples of Selecting Output Using the WHERE= Option

```
ods html file="program4_17.html";
proc document name=mylib.quickstart(read);
/* searching for a path name */
title "LIST 1: Output Objects Pertaining to Vehicle Weight";
list ^(where=(_path_ ? "Weight")) / levels=all;
run;

title "LIST 2: Listing of Tables";
list ^(where=(_type_ = "Table")) / levels=all;
run;

title "LIST 3: All Tables and Graphs";
list ^ (where=(_type_ in ('Table','Graph'))) / levels=all by groups;
run;

title "LIST 4: Using BY-Group variable as a search parameter";
list ^ (where=(_type_ = "Table" and Origin="Asia"))/ levels=all
      by groups;
run;
quit;
ods html close;
```

Output 4.13 shows the output from LIST 1. The others are conceptually similar. With the LIST statement and the WHERE statement, you can create listings of many useful combinations of output. These listings can serve as a table of contents when you replay the actual output in the next chapter, “The REPLAY Statement.” LIST statements are also useful as a previewer for your output. Armed with a table of contents, you can feel more confident that your reports will contain precisely the exhibits that you want, and no other.

Output 4.13 shows the table for LIST 3.

Output 4.13: Results of Using a WHERE Clause to Get a Subset of Tables

LIST 1: Output Objects Pertaining to Vehicle Weight

Listing of: \Mylib.Quickstart\where=(_path_ contains 'Weight')

Order by: Insertion

Number of levels: All

| Obs | Path | Type |
|-----|--|-------|
| 1 | \Univariate#1\ByGroup1#1\Weight#1 | Dir |
| 2 | \Univariate#1\ByGroup1#1\Weight#1\Moments#1 | Table |
| 3 | \Univariate#1\ByGroup1#1\Weight#1\BasicMeasures#1 | Table |
| 4 | \Univariate#1\ByGroup1#1\Weight#1\TestsForLocation#1 | Table |
| 5 | \Univariate#1\ByGroup1#1\Weight#1\Quantiles#1 | Table |
| 6 | \Univariate#1\ByGroup1#1\Weight#1\ExtremeObs#1 | Table |
| 7 | \Univariate#1\ByGroup2#1\Weight#1 | Dir |
| 8 | \Univariate#1\ByGroup2#1\Weight#1\Moments#1 | Table |
| 9 | \Univariate#1\ByGroup2#1\Weight#1\BasicMeasures#1 | Table |

Interlude

Before going further with the WHERE= statement, it's time to stop to see that what you have accomplished would be very difficult to do without the DOCUMENT procedure. You have stored all output in the document rather than print it out immediately, and you might have wondered why make an additional step. Hopefully, Program 4.15 has convinced you of the merits of this approach. The DOCUMENT procedure permitted you to extract a list of outputs of interest, even when SAS did not store them or print them adjacent to one another. Furthermore, you can get this list (and the accompanying graphs) at any time, without having to rerun the original analysis. With the WHERE= statement, you have all your output at your fingertips, ready to be printed in any subset or any order you want.

In the next chapter, this very same WHERE= clause (modified slightly) will let you replay all of the graphs in the document.

You can use any path as the starting point for your search. Program 4.18 shows how to get the listing of all CDFPLOT objects in the UNIVARIATE folder in the TEMP2 document. The TEMP2 document is defined in Program 4.8.

Program 4.18: Using a Specific Path

```
ods html file="program4_18.html";
title "Using WHERE statements To Find Cumulative Distribution Plots";
proc document name=temp2;
  list univariate#1(where=(_name_='CDFPlot')) / levels=all bygroups;
  run;
quit;
ods html close;
```

Output 4.14 is a listing of all the graphs that meet the criteria. It also makes a helpful table of contents, since all of the BY groups are listed, so you know which graph is which.

Output 4.14: Using the WHERE= Option to Find a Specific Exhibit

| Using WHERE statements to find graphical CDF Plots | | | | |
|---|--|-------|--------|--------|
| Listing of: Work.Temp2\Univariate#1(where=(_name_='CDFPlot')) | | | | |
| Order by: Insertion | | | | |
| Number of levels: All | | | | |
| Obs | Path | Type | Origin | Type |
| 1 | \Univariate#1\ByGroup1#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | Asia | SUV |
| 2 | \Univariate#1\ByGroup2#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | Asia | Sedan |
| 3 | \Univariate#1\ByGroup3#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | Asia | Sports |
| 4 | \Univariate#1\ByGroup4#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | Asia | Truck |
| 5 | \Univariate#1\ByGroup5#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | Asia | Wagon |
| 6 | \Univariate#1\ByGroup6#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | Europe | SUV |
| 7 | \Univariate#1\ByGroup7#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | Europe | Sedan |
| 8 | \Univariate#1\ByGroup8#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | Europe | Sports |
| 9 | \Univariate#1\ByGroup9#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | Europe | Wagon |
| 10 | \Univariate#1\ByGroup10#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | USA | SUV |
| 11 | \Univariate#1\ByGroup11#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | USA | Sedan |
| 12 | \Univariate#1\ByGroup12#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | USA | Sports |
| 13 | \Univariate#1\ByGroup13#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | USA | Truck |
| 14 | \Univariate#1\ByGroup14#1\MSRP#1\CDFPlot#1\CDFPlot#1 | Graph | USA | Wagon |

Debugging WHERE= Clauses

You saw earlier that if the LEVELS=ALL option is removed from the LIST statement with the BYGROUP option, the statement might produce no output. Similarly, when using WHERE= clauses, it is safer to use the LEVELS=ALL option in order to have the entire document included in the search.

In Program 4.19, the user wanted a list of all directories pertaining to the WEIGHT variable in WORK.CARS (this data set was produced by Program 1.1 in the quick-start guide).

Program 4.19: When WHERE= Statements Fail

```
proc document name=mylib.quickstart;
/* type here refers to a data set variable, NOT the
/* document variable object type. */
/* the lack of underscore means data set variables */
list univariate(where=(type='Weight' ①));
run;
quit;
```

Program 4.20 produces no output. To see the problem, remove the WHERE= clause ① to see the unrestricted directory of the pathname listed. This is demonstrated in Program 4.20.

Program 4.20: Debug by Removing the WHERE= Clause

```
proc document name=mylib.quickstart;
list univariate / levels=1;
run;
```

Output 4.15: Unrestricted Output

| Debugging | | |
|--|------------|------|
| Listing of: \Mylib.Quickstart\Univariate#1 | | |
| Order by: Insertion | | |
| Number of levels: 1 | | |
| Obs | Path | Type |
| 1 | ByGroup1#1 | Dir |
| 2 | ByGroup2#1 | Dir |
| 3 | ByGroup3#1 | Dir |
| 4 | ByGroup4#1 | Dir |
| 5 | ByGroup5#1 | Dir |
| 6 | ByGroup6#1 | Dir |

Without the WHERE= clause, the LIST statement shows that there are no output objects at this level, and therefore nothing to test. Data set variables can be tested in WHERE clauses only on output objects.

If you wanted to get a list of all output objects pertaining to the variable WEIGHT, one way to do this would be Program 4.21.

Program 4.21: Corrected Version

```
proc document name=mylib.quickstart;
list univariate(where=(ucase(_path_) ? 'WEIGHT')) / levels=all;
run;
quit;
```

Another way of thinking of this is that the `_path_` variable will be created only to the depth that is specified by the LEVELS= command, and LEVELS=1 did not specify a level deep enough to see any output objects.

Use caution with the WHERE= statement. To ensure that your clause searches the entire document tree, it is usually wise to specify LEVELS=ALL in conjunction with WHERE= clauses.

The WHERE= Clause and ODS Names

In Chapter 2 you learned that every object created by ODS has a name. This ODS name can be found at the rightmost part of each path, and can be tested for directly without having to use a Contains operator on the entire path. Program 4.22 lists all entries that are named or contain the tokens FitStatistics Or ParameterEstimates.

Program 4.22: Two Equivalent Ways of Searching for an Output Object

```
proc document name=my.lib.firstdoc;
/* tomato */
list reg#2(where=_NAME_in ('FitStatistics','ParameterEstimates'));
/* tomahtoe */
list reg#2(where=(_path ? 'FitStatistics' || path ?
'ParameterEstimates')));
run;
quit;
```

When you test for `_NAME_` in this code, you are really testing just the rightmost component of the pathname. Either of the two versions above finds the output objects named `Fitstatistics` and `ParameterEstimates`. However, testing `_NAME_` directly is syntactically simpler.

Comparison with the ODS SELECT and EXCLUDE Statements

Program 4.20 showed another way of selecting and excluding output. Recall that Chapter 3, “The ODS DOCUMENT Statement,” mentioned the following statement:

```
ODS DOCUMENT SELECT fitstatistics parameterestimates;
```

When executed in open code while running the REG procedure, this statement restricts output to the document, and only output with ODS names of Fitstatistics or ParameterEstimates are stored. Once the REG procedure had run, you had to remember to reset the SELECT and EXCLUDE lists to ensure that all output from other procedures would be included.

Rather than deal with the burden of remembering which SELECT and EXCLUDE lists were in effect, why not just save *all output* to the document and later restrict output using the WHERE= statements? The replay with WHERE= clauses produces the same output, yet no work is lost.

LABELPATH - Proceed at Your Own Risk

The _LABELPATH_ document variable is quite handy, but be aware of its limitations within and across locales.

ODS documents are locale-sensitive. If a document is created and replayed within a single locale, this should pose no problem. However, if you pass your document to a team-member in another locale, they may not always be able to search reliably on it.



As cited in *Output Delivery System: The Basics and Beyond*, labels in _LABELPATH_ may vary by country and locale. For example, the `moments` object of the UNIVARIATE procedure will be named something in French, and something else in Mandarin. But the ODS name of `moments` will be in English, regardless of locale. Therefore, the author recommends that if you plan to share your documents with SAS users in different locales, use _PATH_ for subsetting rather than the _LABELPATH_. The _LABEL_ variable is discussed in the section with SETLABEL statement, later in this chapter. Program 4.23 demonstrates subsetting on the _LABELPATH_ variable.

Program 4.23: Searching on _LABELPATH_

```
ods document name=explode(write);
/* write mode used because the aim is to start
   From scratch with an empty document */
proc univariate data=sashelp.retail;
run;
ods document close;

proc document name=explode;
list ^ (where=(_labelpath_ ? "The Univariate Procedure"));
run;
quit;
```

The reason to avoid searching on labels is that labels are not guaranteed to be the same in different locales. However, if all of the people reading your document are in the same country, then there should be little problem.

For more information, and for examples of WHERE= clause processing, see the ODS Document Tip Sheet, which appears in the Knowledge Base in the Focus Areas section of the SAS Support Web site. The tip sheet also provides a complete list of all the special document variables for WHERE= clause processing.

The SETLABEL Statement

Now that you have learned how to navigate through a large document listing using WHERE statements, you have won only half the battle. The other half is remembering WHY the output was saved at all! SETLABEL can help you by giving you space to provide comments about why an output folder was created, or how one report's output might differ from another's.

The SETLABEL statement makes your lists easier to read by providing a descriptive label. Labels follow the same rules as labels in other areas of SAS. The limit on length is 256 characters.

The labels that are created by the SETLABEL statement can be used as input to the WHERE= option statements by using the _LABEL_ document variable.

The Syntax of the SETLABEL Statement

```
SETLABEL path quoted-string;
```

The quoted string can be either single or double quotes, so that you can use macro variable substitution, if you want.

Program 4.24 below assigns some more descriptive labels and displays a listing.

Program 4.24: Assigning Labels

```
ods html file="program4_24.html";
proc document name=mylib.firstdoc;
  setlabel reg#1 'Model is weight = height + error, by sex.';
  setlabel reg#2 'Same Model as Reg#1 but includes all tables';
  list / details ① levels=1;
  run;
ods html close;
```

- ① You must use the DETAILS option to see labels in listings. For brevity, some columns are omitted from the output. If you are wondering how to modify which columns appear in the document listing, note that the macro %thin_list creates a detailed listing like the one in Output 4.16. %restore_list restores the original columns. The code for these macros is included in the Example Code and Data on the author page for this book.

One reason to use the DOCUMENT procedure instead of some external method, such as spreadsheet files, to manage output is that most operating systems do not automatically provide labels of directories or of individual outputs.

Output 4.16: Detailed Listing of the Document with Labels

Labeling Directories

Listing of: \Mylib.Firstdoc

Order by: Insertion

Number of levels: 1

| Path | Type | Size in Bytes | Template | Label |
|---------------|------|------------------|----------|---|
| \Univariate#1 | Dir | | | The Univariate Procedure |
| \Freq#1 | Dir | | | The Freq Procedure |
| \special#1 | Dir | | | All analysis pertaining to SASHELP.AIR |
| \Reg#1 | Dir | | | Model is weight = height + error, by sex. |
| \Reg#2 | Dir | | | Same Model as Reg#1 no grouping by sex |

Using SETLABEL= to Label an Entire Document

Program 4.25 demonstrates that SETLABEL can also label an entire document, just as the LABEL= option in the DOC statement or the LABEL= option in the PROC DOCUMENT statement.

Absolute and Relative Pathnames

This example also provides a look at *absolute pathnames*. An *absolute pathname* is just a backslash, followed by the document name, followed by a pathname within that document. The absolute pathname `\mylib.firstdoc\` specifies the root directory of MYLIB.FIRSTDOC. If the current document is specified by the NAME= option to the DOCUMENT procedure, the root directory can be written as `\`. In short, any path that begins with a backslash is an absolute pathname.

A relative pathname does not have a leading backslash. Therefore, it is interpreted relative to the current directory. If the current directory is `\snapshot#1` and the relative path is `reports#1`, then the absolute pathname equivalent is `\snapshot#1\reports#1`.

One application of absolute pathnames is to label a document, even when it is not the current document or when there is no current document.

Program 4.25: SETLABEL Can also Label a Document as a Whole

```
ods html file='program4_25.html';
proc document;
/* no current document needed for this example */
setlabel \mylib.firstdoc\ 'First Document: Analysis of SASHELP.CLASS';
setlabel \mylib.quickstart\ 'Quick Start : Analysis of SASHELP.CARS';
doc;
run;
quit;
ods html close;
```

Important: Labels Are Not Titles



Labels are metadata. Do not confuse object labels, the ones that are created with SETLABEL, with titles in SAS TITLE_n statements. These labels *never* appear in TITLE_n titles.

Although path labels, or labels for short, are metadata, they can make an appearance in some file formats that display metadata. One example is the PDF destination. In that context, labels can appear as bookmarks to locations within the PDF.

You can search based on the labels that you created with the SETLABEL statement. Program the WHERE clause to search through the document, as Program 4.26 shows:

Program 4.26: Searching by Label

```
ods html file="program4_26.html";
proc document name=mylib.firstdoc;
  setlabel freq 'First Attempt';
  list ^(where=(_label_='First Attempt'));
run;
quit;
```

You might find this easier than searching on ODS names, since the labels can reflect your preferences, or your team's naming conventions.

Output 4.17: All Objects with a Specific Label

| Searching by Label | | | | |
|--|------|------------------|----------|---------------|
| Listing of: \Mylib.Firstdoc\(^{where=(_label_ = 'First Attempt')}) | | | | |
| Order by: Insertion | | | | |
| Number of levels: 1 | | | | |
| Path | Type | Size in Bytes | Template | Label |
| \Freq#1 | Dir | | | First Attempt |

The listing shows only the folder that has the label named First Attempt. This does not include its subfolders. The label of a folder *does not* in any way affect the labels of its subfolders, by design. The detailed listing of the FREQ#1 folder, shown in Program 4.27 demonstrates this.

Program 4.27: Labels of Folders and Subfolders are Independent of One Another

```
/* program 4.27 */
ods html file="program4_26.html";
title "Labels do not cascade";
proc document name=mylib.firstdoc;
  setlabel freq 'First Attempt';
  list ^(where=(_path_ ? 'Freq')) / levels=all details;
run;
quit;
ods html close;
```

Output 4.18 shows the output of the LIST statement.

Output 4.18: Labels of Folders and Subfolders

Listing of: \Mylib.Firstdoc\where=(_path_ contains 'Freq'))

Order by: Insertion

Number of levels: All

| Path | Type | Size in Bytes | Template | Label |
|--------------------------------|-------|------------------|-----------------------|---------------------|
| \Freq#1 | Dir | | | First Attempt |
| \Freq#1\Table1#1 | Dir | | | Table Sex |
| \Freq#1\Table1#1\OneWayFreqs#1 | Table | 423 | Base.Freq.OneWayFreqs | One-Way Frequencies |

Being able to create your own labels and search for them with a WHERE= option gives you a powerful tool for organizing your SAS procedure output.

Summary

This chapter showed you how to invoke the DOCUMENT procedure to see what SAS output is stored within it. The author believes that managing your stored SAS output is as important as replaying it. Output should be easy to find, and it should be equally easy to determine the *meaning* and purpose of your stored output without having to replay it. The mantra of `proc document;replay;run;quit;` won't help you if your printed output is nearly a thousand pages long.

Therefore, it is necessary to manage the size of your printed output. A prerequisite to this is to be able to list the output that you plan to print. This is why the chapter on LIST appears before the chapter on REPLAY.

You can manage the size of the listing in several ways. You can list only a specific folder. You can select a subset of folder locations by either using a WHERE statement. Finally, you can control how many levels of folders to display with the LEVELS= option.

You can use the document lists to preview the output that you will eventually replay, or you can print out your LIST statement output to use as a table of contents and have it be part of your report.

Finally, the lists, with their ODS names and many sequence numbers, would be hard to read without some easy-to-read labels. The SETLABEL command will label any path. You must use the DETAILS option to see the labels in the document listing.

The LIST statement will prove to be useful in obtaining a preview of what will be replayed. You will learn about the REPLAY command in the next chapter.

Answer to Quick Check-in: All sequence numbers are #1. The sequence number comes after the hash sign (#). Any numbers before are part of the folder's name. For instance BYGROUP31415#1 has a name of BYGROUP31415 and a sequence number of 1.

Chapter 5: The REPLAY Statement

[Introduction](#)

[How Replay Works](#)

[REPLAY Statement Syntax](#)

[Replaying to All Open Destinations](#)

[The DEST= Option](#)

[Replaying to Multiple Destinations](#)

[Replaying in a Different Order](#)

[The LEVELS= Option](#)

[Overriding Titles with the ACTIVETITLE and ACTIVEFOOTN Options](#)

[Replaying Using WHERE= Clauses](#)

[Replaying Subsets of Output Objects](#)

[Example: Reordering BY-Grouped Output](#)

[Replaying Output with Temporary Formats](#)

[Temporary or Permanent Formats?](#)

[Summary](#)

Introduction

The REPLAY statement re-creates your original output in a variety of styles and layouts. The REPLAY statement sends output from a document item store to an ODS destination without your having to rerun the original SAS analysis.

Although not part of the REPLAY discussion proper, the topic of SAS formats and how they interact with document replay is explored in this chapter because users are concerned about output replaying faithfully compared with its original. Formats are an important part of that discussion. The document stores temporary formats, and rebuilds them in the WORK library each time the document is replayed.

It is worth restating that despite all the fancy bells and whistles, using the DOCUMENT procedure is easy as

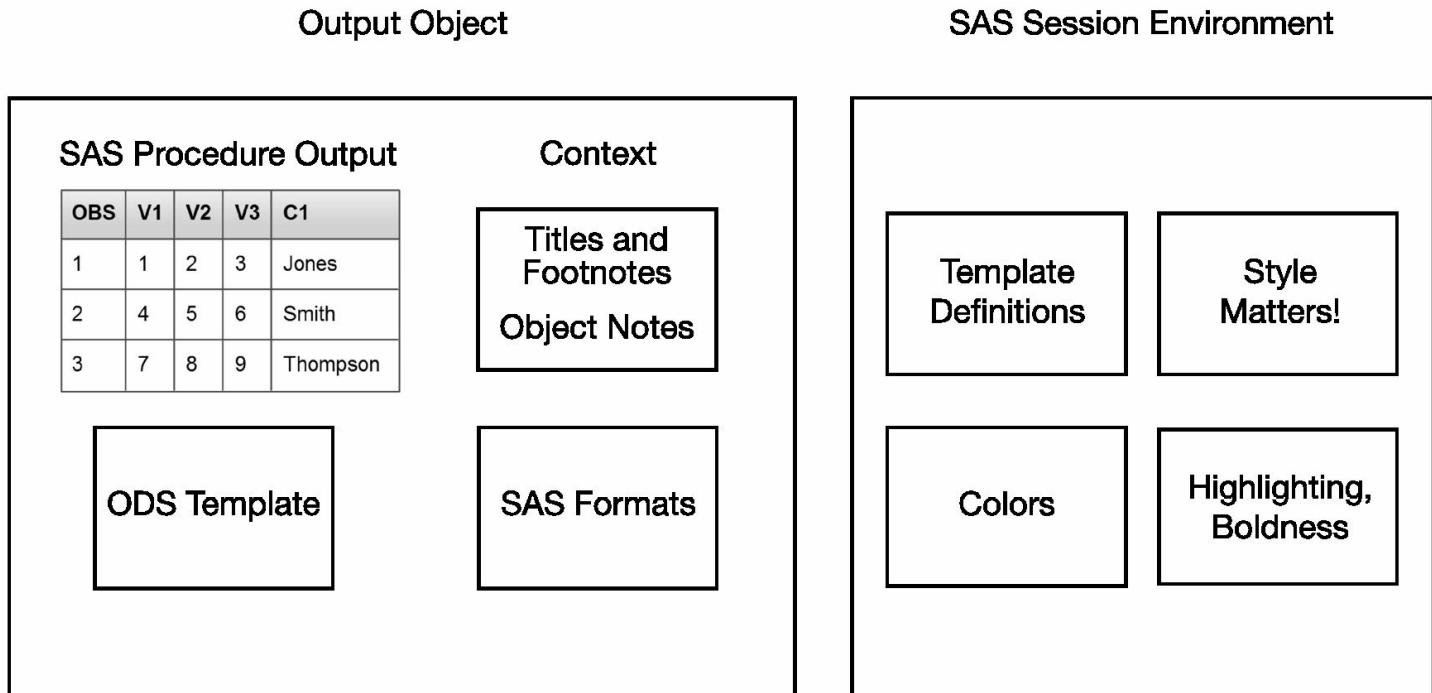
```
"proc document; replay; run; quit".
```

How Replay Works

In order to replay an output object, ODS must assemble several ingredients. One way to envision the replay process is to use the model in Figure 5.1. It is true that there are a few SAS procedures that do not follow this model exactly. However, the point is to envision a process that works most of the time so that you can see the role each aspect plays. For example, formats are not literally stored with the output object. The appearance of replayed output depends on two broad factors: what is stored in the document itself and what is specified from the current SAS session.

Starting with the output object, the model shows that each output object consists of data that is stored in tabular format: the template that is used, titles, footnotes, page-breaks, and SAS formats. The template can either be stored internally with the document, or it can be stored in a separate part of SAS. The ODS document stores *preformatted data*. That is, it stores data without the style elements in the righthand box in Figure 5.1.

Figure 5.1: Information REPLAY Needs to Rebuild Output



Preformatted data is like freeze-dried vegetables. I acknowledge Cynthia Zender for providing the following tasty analogy: What you do to turn it into haute cuisine depends on what else you add to it when you remove it from the freezer. The freezer is your document, and the spices and additional ingredients are your style elements. You could just thaw it out and serve it; that corresponds to a straightforward replay. Nevertheless, you could also add other sauces, or combine it with other dishes. That corresponds to how the templates and styles add to the appearance of the document at replay time.

The key to successfully replaying documents with the desired appearance is to understand which elements of the output are stored in the document itself, and which elements are based on the state of the system at replay time. To fill in the details that are needed to create formatted output, ODS uses the templates and styles that are in effect at the time the REPLAY statement is executed. Typically, for a beginning user of the document, these are the same settings as were used when the output was originally sent to the DOCUMENT destination. Thus, the output would appear just as

it would if it were printed directly to the destination.

For example, you could use a FORMAT procedure statement to change an active format. See the section about temporary formats later in this chapter. Or, you could remove a column from a table by editing the procedure template's column statement.

This chapter shows an example of changing the appearance of the replayed output by changing a format and specifying a different style. Modifying a procedure's template is a more advanced example, and is deferred until Chapter 11, "Working with Templates." If you leave templates and styles alone, the output will replay as it was when it was first created.

REPLAY Statement Syntax

Here is the syntax of the REPLAY statement:

```
REPLAY pathname1 [(where-expression)],  
  pathname2(where-expression) ,  
  pathname3(where-expression) , ...  
 / options
```

The REPLAY statement has the following options:

```
DEST=destination | (destination1 destination2 ...destination)  
LEVELS=N|ALL  
[ACTIVEFOOTN | ACTFOOTN ]  
[ACTIVETITLE | ACTITLE];
```

Each option is discussed in turn.

Replaying to All Open Destinations

If you have not yet built the `MYLIB.FIRSTDOC` document, it can be created by running Program 3.1 followed by Program 3.4. The combined program is available on the SAS Press author page for this book at support.sas.com/tuchman. You have already seen a replay of the entire document using a REPLAY statement with no arguments. Therefore, the first example in this chapter shows you how to select a single object for replay.

Program 5.1: Default Behavior of the REPLAY Statement Is to Replay to all Open Destinations

```
ods pdf file="program5_1.pdf";
ods html(id=bluish) file="program5_1.html"
  newfile=none style=htmlblue ❶ ❷;
ods html(id=serious) file="chap5_simple.html" style=statistical;
ods tagsets.excelxp file="chap5.xml";
ods listing close;
```

```
proc document name=mylib.firstdoc(read); ❸
  replay univariate#1/by group 1#1\height\moments; ❹
run; ❺
quit;

ods _all_close; ❻
```

This example shows quite a number of destinations open at once. An ODS document lets you experiment with different destinations as well as styles, all without rerunning the original code. This example also features two HTML destinations that are open at the same time with different settings for each.

- ❶ NEWFILE=NONE sends all output to the HTML file that is specified, and does not try to create a separate file for each procedure.

The STYLE= statement takes as an argument an HTML style. You can write your own styles or use the ones provided with SAS. In order to concentrate on the document rather than ODS style code, attention is restricted to styles that come with a standard installation of SAS. If you are advanced in ODS and know about writing your own styles, and thus can ensure that your styles are correctly placed on the ODS PATH, feel free to use them instead of the standard SAS styles. Custom styles enhance your power with the ODS document.

In this example, notice that the change of a single keyword, notably the style name after the STYLE= option, can enhance your document's appearance. You do not need an ODS document to use styles, of course. However, the ODS Document Facility makes it easier to try several styles to find the presentation statement that fits your message.

ODS styles that this book refers to are listed in the appendix.

- ❷ This statement completes the opening of all of the ODS destinations for this program. The ODS destinations that are used here include two HTML files, each with its own style, one PDF destination, and one spreadsheet destination using the TAGSETS.EXCELXP destination. You might have two different versions of the same ODS destination open at the same time, provided they are distinguished by this ID= statement. Notice that the ID= value takes no quotation marks.
- ❸ Open the document `mylib.firstdoc` in Read mode. This prohibits making any changes to the document.

- The REPLAY statement will now replay all of the output to all open destinations. Notice how the DOCUMENT procedure is able to use the same table to create differently styled versions of the same output to different destinations.
- The RUN statement is critical. Without the RUN statement, the example produces no output! To create output, the DOCUMENT procedure needs to have a complete RUN group. RUN groups were discussed in the previous chapter.

```
ods <destination> FILE=file;
proc document;
list;
run;

replay;
run; /* both run statements are needed */
ODS Document close;
```

- The ODS_ALL_CLOSE; statement closes all output, and replaces three individual CLOSE statements. Submit the ODS LISTING statement after your code to reopen the listing destination, if desired.

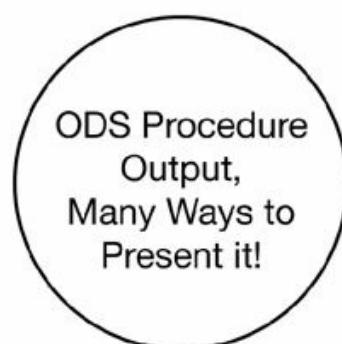
Output 5.1 shows the results.

Output 5.1: Several Presentations of a Single Table

PDF

| The UNIVARIATE Procedure | | | |
|--------------------------|------------|------------------|------------|
| Variable: Height | | | |
| Sex=F | | | |
| Moments | | | |
| N | 9 | Sum Weights | 9 |
| Mean | 60.5888889 | Sum Observations | 545.3 |
| Std Deviation | 5.01832752 | Variance | 25.1836111 |
| Skewness | -0.7238643 | Kurtosis | -0.3464949 |
| Uncorrected SS | 33240.59 | Corrected SS | 201.468889 |
| Coeff Variation | 8.28258714 | Std Error Mean | 1.67277584 |

| Sex=F | | | |
|-----------------|------------|------------------|------------|
| Moments | | | |
| N | 9 | Sum Weights | 9 |
| Mean | 60.5888889 | Sum Observations | 545.3 |
| Std Deviation | 5.01832752 | Variance | 25.1836111 |
| Skewness | -0.7238643 | Kurtosis | -0.3464949 |
| Uncorrected SS | 33240.59 | Corrected SS | 201.468889 |
| Coeff Variation | 8.28258714 | Std Error Mean | 1.67277584 |



| The UNIVARIATE Procedure | | | |
|--------------------------|------------|------------------|------------|
| Variable: Height | | | |
| Sex=F | | | |
| Moments | | | |
| N | 9 | Sum Weights | 9 |
| Mean | 60.5888889 | Sum Observations | 545.3 |
| Std Deviation | 5.01832752 | Variance | 25.1836111 |
| Skewness | -0.7238643 | Kurtosis | -0.3464949 |
| Uncorrected SS | 33240.59 | Corrected SS | 201.468889 |
| Coeff Variation | 8.28258714 | Std Error Mean | 1.67277584 |

Spreadsheet

| The UNIVARIATE Procedure | | | |
|--------------------------|------------|------------------|------------|
| Variable: Height | | | |
| Sex=F | | | |
| Moments | | | |
| N | 9 | Sum Weights | 9 |
| Mean | 60.5888889 | Sum Observations | 545.3 |
| Std Deviation | 5.01832752 | Variance | 25.1836111 |
| Skewness | -0.7238643 | Kurtosis | -0.3464949 |
| Uncorrected SS | 33240.59 | Corrected SS | 201.468889 |
| Coeff Variation | 8.28258714 | Std Error Mean | 1.67277584 |

HTML
Blue Style of HTML

Statistical Style of HTML

The DEST= Option

When you want finer control over which output is sent to which destination, use the DEST= option. Consider the following example, which shows the DEST option using `MYLIB.FIRSTDOC` as the example document. It shows how the same output can be rendered in different output formats.

Replaying to Multiple Destinations

If you use the DEST= option instead of printing to all available destinations, you might specify multiple destinations for the same output by enclosing them in parentheses with spaces between them. For example, Program 5.2 sends the output from the REG procedure to an XML spreadsheet and to the HTML destination. This is a good time to mention that if there are graphs in your output, but your destination does not display graphs, the OUTPUT destination will ignore it and wait for the next object that it can display.

Program 5.2: Illustrating Different Output Formats

```
/* same ODS destinations as program 5.1 */  
proc document name=mylib.firstdoc;  
  replay reg / dest=(tagsets.excelxp html(id=bluish));  
  run;  
quit;
```

Output 5.2 shows the two versions of the output as they appear in their respective destinations.

Output 5.2: Specifying Two Destinations Out of Several Possible Open Destinations

The screenshot displays two outputs side-by-side. On the left is an Excel spreadsheet showing two rows of data: 'Number of Observations Read' and 'Number of Observations Used', both with the value 19. Below this is an Analysis of Variance table for a REG procedure. The table includes columns for Source, DF, Sum of Squares, Mean Square, F Value, and Pr > F. The results show one model term with 1 DF and 7193.24912 Sum of Squares, and 17 Error terms with 2142.48772 Sum of Squares. The total has 18 DF and 9335.73684 Sum of Squares. On the right is an HTML page titled 'The REG Procedure Model: MODEL1 Dependent Variable: Weight'. It contains the same two rows of data and the Analysis of Variance table. Below the table are sections for Root MSE (11.22625), R-Square (0.7705), Dependent Mean (100.02632), Adj R-Sq (0.7570), and Coeff Var (11.22330). At the bottom is a Parameter Estimates table with columns for Variable, DF, Parameter Estimate, Standard Error, t Value, and Pr > |t|. The Intercept has a parameter estimate of -143.02692 and a t value of -4.43, while Height has a parameter estimate of 3.89903 and a t value of 7.55.

If a destination is open, and no output was sent to it, you might see this warning in the log as shown in Display 5.1. In this case, the PDF destination was open but received no output. However, there was no problem with your RUN group in this instance.

Display 5.1: Log Warning

```
3657  
3658 ods _all_ close;  
NOTE: ODS PDF printed no output.  
(This sometimes results from failing to place a RUN statement before the ODS PDF CLOSE statement.)
```

Replaying in a Different Order

The REPLAY statement permits the sending of output to multiple pathnames in a single command. For example, you could use this approach to reorder or subset your output. Program 5.3 shows a document replay and plays information in a different order than it was stored.

Program 5.3: Changing the Order of Replay

```
ods pdf file='program5_3.pdf';
proc document name=mylib.quickstart;
  replay sgpanel, means / dest=pdf;
  run;
quit;
ods pdf close;
```

The output now shows the variable listing from the CONTENTS procedure being printed after that of the MEANS procedure, even though it was stored at an earlier position in the document.

Output 5.3: Printing in a Different Order

The screenshot shows the SAS Output window with two tables displayed side-by-side. On the left, a table provides detailed information for a 'Wagon' model, including MSRP, Invoice, EngineSize, Cylinders, Horsepower, MPG_City, MPG_Highway, Weight, and Wheelbase Length. On the right, an 'Alphabetic List of Variables and Attributes' table lists variables by their numerical position (#), variable name, type, length (Len), format, and label. The variables listed are Cylinders, DriveTrain, EngineSize, Horsepower, and Invoice.

| # | Variable | Type | Len | Format | Label |
|----|------------|------|-----|----------|-----------------|
| 9 | Cylinders | Num | 8 | | |
| 5 | DriveTrain | Char | 5 | | |
| 8 | EngineSize | Num | 8 | | Engine Size (L) |
| 10 | Horsepower | Num | 8 | | |
| 7 | Invoice | Num | 8 | DOLLARS. | |

The LEVELS= Option

The LEVELS= option becomes more useful after you have learned the MOVE option.

Therefore, you will see examples of the LEVELS statement in Chapter 6, “Managing Folders,” after the discussion of COPY TO and MOVE TO statements.

This chapter covers routinely printing all output in a folder, which is the most common application. For most replay applications, it is necessary only to be aware of LEVELS=ALL, which is the default.

Overriding Titles with the ACTIVETITLE and ACTIVEFOOTN Options

The ACTIVETITLE and ACTIVEFOOTN options instruct SAS to use the titles and footnotes from the current TITLE and FOOTNOTE statements rather than those stored with the output in the ODS document.

These options will become more useful after the discussion of annotating output with the OBANOTE statement. Examples are not shown in this chapter, but they are covered in Chapter 7, “Customizing Output.”

Replaying Using WHERE= Clauses

As with the LIST statement, you can use the WHERE= option when specifying document paths to subset ODS output. The syntax of the WHERE= option was discussed in Chapter 4, “Listing Documents Using the DOCUMENT Procedure.” To specify which paths are of interest, special document variables characterized by leading and trailing underscores were also listed in that chapter. You can use these WHERE= clauses to select output for the REPLAY statement. Program 5.4 shows how to combine the REPLAY and WHERE options on the pathname.

Program 5.4: Replying Using the WHERE Option

```
ods html(id=serious) file='program5_4.html' style=analysis;
ods pdf file='program5_4.pdf';
ods tagsets.excelxp file='program5_4.xml';
proc document name=my.lib.firstdoc(read);
  replay reg / dest=tagsets.excelxp;
run;

replay univariate(where=(_name_='Moments'))
  / dest=html;
run;
replay univariate(where=(_path_ ? 'Height'))
  / dest=html;
run;
replay reg(where=(sex='F')) / dest=pdf;
run;
quit;
ods _all_close;
```

Notice that there is more control over which outputs go to which destinations, compared with not having a WHERE option at all.

In Program 5.4, the output from the REG procedure is sent to the spreadsheet destination, the Moments output from the UNIVARIATE procedure is sent to the HTML destination with ID=bluish, the output for the Height variable is sent exclusively to the HTML destination with ID=serious, and the REG procedure output that is restricted to Sex=F is sent to the PDF destination. Note the use of the BY variable SEX in the WHERE option. Only a variable that is used as a BY-variable in the original analysis can be used in the WHERE option.

There is an exception to this. If the object that is being replayed is a single output object, you can use any of the variables in that output object as a variable in the WHERE= option. Program 5.8 later in this chapter demonstrates this.

Replaying Subsets of Output Objects

SAS 9.3 expanded the reach of the WHERE= option by allowing replaying of any part of an output object. The following are new keywords: _MIN_, _MAX_, and _OBS_. These are intended for use only in the WHERE= expression. These new keywords are defined in Table 5.1. Furthermore, these can be used only in the WHERE= option for an output object.

Table 5.1: Row Subsetting Variables

| | |
|---------------|--|
| _MIN_ | A synonym for the first observation, namely the top row in the output as it appears. _MIN_ always equals 1. |
| _MAX_ | Number of observations in the output object. |
| _OBS_ | Refers to the observation number that you want to print. |
| Variable name | A variable name in the stored output table. For example, BY-group variables used in producing the output table are in this category. |

A visual guide might make this clearer. Display 5.2 shows the definitions in the context of an actual table.

Display 5.2: Row Subsetting Illustrations

| Parts of a Table | |
|-------------------------|--|
| _MIN_=1 _MAX_= 6 | |
| _OBS_= 1 | Sex F M Height Height Mean Mean |
| _OBS_= 3 | Age |
| _OBS_= _MAX_ | 11 51.30 57.50 12 58.05 60.37 13 60.90 62.50 14 63.55 66.25 15 64.50 66.75 16 . 72.00 |

Essentially, subsetting rows of output objects can be done completely analogously with the way you are accustomed to subsetting SAS tables. Note, however, that the dropping of columns is not supported. In order to use this technique, you need to be referring to an output object, and not to a directory. In Program 5.4, the output objects to be replayed are the quantiles for Height from the UNIVARIATE procedure. There are several ways to obtain these pathnames; probably the easiest for short jobs is to use the LIST statement, and then cut and paste the results into your code. Program 5.5 shows how these pathnames can be obtained.

Program 5.5: Replying the First Row of an Output Object

```
proc document name=my.lib.firstdoc;
  title "Replaying Part of an ODS Table";
  replay univariate#1/by group 1#1\Height#1\quantiles
    (where=(_obs_= 1)) / activetitle;
  replay univariate#1/by group 2#1\Height#1\quantiles
```

```
(where=(_obs_= 1));  
run;  
quit;
```

This prints the first row of the height quantile tables for SEX=F and SEX=M. Quantile tables are laid out so that the maximum is the first row, which means it prints the maximum. Output 5.4 shows what this looks like.

Output 5.4: Replying the First Row

The image shows two vertically stacked SAS output panels. Both panels have a header 'The SAS System' and a sub-header 'The UNIVARIATE Procedure Variable: Height'. The top panel is for 'Sex=F' and the bottom panel is for 'Sex=M'. Each panel contains a table titled 'Quantiles (Definition 5)' with two columns: 'Quantile' and 'Estimate'. In the Sex=F panel, the '100% Max' quantile has an estimate of 66.5. In the Sex=M panel, the '100% Max' quantile has an estimate of 72.

| Quantiles (Definition 5) | |
|--------------------------|----------|
| Quantile | Estimate |
| 100% Max | 66.5 |

| Quantiles (Definition 5) | |
|--------------------------|----------|
| Quantile | Estimate |
| 100% Max | 72 |

If you do not want to remember the entire pathname, you could also try listing the path using WHERE= options, as demonstrated in Program 5.6. Program 5.6, properly speaking, belongs in the discussion with the LIST statement, but a common issue when replaying is to make sure you have valid objects to replay. Using the LIST statement helps you pinpoint exactly what the document would replay, when you replace the LIST statement with a REPLAY statement. For the time being, you can cut and paste from the LIST output to use the pathnames in your DOCUMENT procedure code.

Program 5.6: Which Path to Replay?

```
ods html file='program5_6.html';  
proc document name=my.lib.firstdoc;  
  title "Replaying Part of an ODS Table";  
  list univariate#1(where=(upcase(_path_) ? 'HEIGHT' and upcase( _name_) =  
    'QUANTILES' and upcase(_type_) ='TABLE')) / levels=all by groups;  
  run;  
quit;  
ods html close;
```

The ODS pathname is case sensitive; use the UPCASE function to take the worry out of searching.

Output 5.5: Using the LIST Statement to Obtain Paths to Output Objects

Listing of: \Mylib.Firstdoc\Univariate#1(where=(UPCASE(_path_) contains 'HEIGHT' and (UPCASE(_name_)='QUANTILES') and (UPCASE(_type_) ='TABLE')))

Order by: Insertion

Number of levels: All

| Obs | Path | Type | Sex |
|-----|---|-------|-----|
| 1 | \Univariate#1\ByGroup1#1\Height#1\Quantiles#1 | Table | F |
| 2 | \Univariate#1\ByGroup2#1\Height#1\Quantiles#1 | Table | M |

Program 5.7 shows another example of how you can use _OBS_, _MAX_, _MIN_ in the WHERE= clause to print out the last five rows of a table.

Program 5.7: Replying the Last Five Rows of a SAS Table

```
proc document name=mylib.firstdoc;
  title "Replaying Part of an ODS Table";
  replay univariate#1/by group 1#1\
    height#1\quantiles#1(where=(_obs_ >= _max_ - 5));
  run;
quit;
```

Output 5.6: The Last Five Rows of an Output Object

| The UNIVARIATE Procedure | |
|---------------------------------|------|
| Variable: Height | |
| Sex=F | |
| Quantiles (Definition 5) | |
| Quantile | |
| 50% Median | 62.5 |
| 25% Q1 | 56.5 |
| 10% | 51.3 |
| 5% | 51.3 |
| 1% | 51.3 |
| 0% Min | 51.3 |

Replaying Parts of a MEANS Table

In Program 5.8, the rows of PROC MEANS output are selectively replayed. Although this example will be covered again using the OUTPUT destination, it is covered here as well. This method is handy because you do not have to go through the step of creating an additional SAS file when all you want is to create a subset of your output. This feature is new in SAS 9.3. For the sake of variety, PDF output is demonstrated in this example, although this feature works equally with all destinations.

Program 5.8: Replying Part of an Output Table

```

ods pdf file='program5_8.pdf';
proc document name=mylib.quickstart;
  replay means\summary(where=(type='SUV'));
  run;
quit;
ods pdf close;

```

The PDF output in Output 5.7 shows that only SUV-related output is produced. Note that Program 5.8 used “`type='SUV'`”. In this instance, the variable TYPE refers to the variable in the output table called type. There is an ODS document variable called `_TYPE_` that can also be used in the WHERE statement.

Output 5.7: Outputting a Subset of MEANS Procedure Output

The MEANS Procedure

| Origin | Type | N Obs | Variable | Label | N Miss | Mean | Maximum | Minimum |
|--------|------|-------|-------------|-----------------|--------|-------------|-------------|-------------|
| Asia | SUV | 25 | MSRP | Engine Size (L) | 0 | 29569.00 | 64800.00 | 17163.00 |
| | | | Invoice | | 0 | 26916.48 | 56455.00 | 16949.00 |
| | | | EngineSize | | 0 | 3.4720000 | 5.6000000 | 2.0000000 |
| | | | Cylinders | | 0 | 6.0000000 | 8.0000000 | 4.0000000 |
| | | | Horsepower | | 0 | 214.1600000 | 325.0000000 | 130.0000000 |
| | | | MPG_City | MPG (City) | 0 | 17.3200000 | 22.0000000 | 13.0000000 |
| | | | MPG_Highway | | 0 | 21.6800000 | 27.0000000 | 17.0000000 |
| | | | Weight | | 0 | 4108.04 | 5590.00 | 3020.00 |
| | | | Wheelbase | | 0 | 108.0400000 | 129.0000000 | 98.0000000 |
| | | | Length | | 0 | 184.8400000 | 208.0000000 | 163.0000000 |
| Europe | SUV | 10 | MSRP | Engine Size (L) | 0 | 48346.00 | 76870.00 | 25995.00 |
| | | | Invoice | | 0 | 44291.30 | 71540.00 | 23969.00 |
| | | | EngineSize | | 0 | 3.9500000 | 5.0000000 | 2.5000000 |
| | | | Cylinders | | 0 | 7.2000000 | 8.0000000 | 6.0000000 |
| | | | Horsepower | | 0 | 263.1000000 | 340.0000000 | 174.0000000 |

Caution Needed When Replaying Parts of Output Objects

This method of subsetting output objects in the REPLAY statement with WHERE= options works best when you have imported the table into the document yourself. The technique can also benefit you in cases where you thoroughly understand the underlying table format that ODS uses to store output. Typically, ODS stores information in a layout that does not resemble the printed output. Consider Program 5.9, which adds output from the MEANS procedure to the document.

Program 5.9: Procedure Code Might Be Stored Differently Than It Appears

```

ods document name=mylib.firstdoc;
proc means data=sashelp.class;
class sex;
run;
ods document close;

ods html file='program5_9.html';
proc document name=mylib.firstdoc;
setlabel means#1 'Demonstration of Subsetting from Chapter 5';
run;

ods output summary=example;
replay means\summary(where=(_obs_=1));
run;

ods output close;
quit;

```

```
proc print data=example;  
run;
```

```
ods html close;
```

The output with no WHERE= option is shown as Output 5.8.

Output 5.8: How to Replay Parts of This Table?

The Original Table

| Sex | N Obs | Variable | N | Mean | Std Dev | Minimum | Maximum |
|-----|-------|----------|----|-------------|------------|------------|-------------|
| F | 9 | Age | 9 | 13.2222222 | 1.3944334 | 11.0000000 | 15.0000000 |
| | | Height | 9 | 60.5888889 | 5.0183275 | 51.3000000 | 66.5000000 |
| | | Weight | 9 | 90.1111111 | 19.3839137 | 50.5000000 | 112.5000000 |
| M | 10 | Age | 10 | 13.4000000 | 1.6465452 | 11.0000000 | 16.0000000 |
| | | Height | 10 | 63.9100000 | 4.9379370 | 57.3000000 | 72.0000000 |
| | | Weight | 10 | 108.9500000 | 22.7271864 | 83.0000000 | 150.0000000 |

Although it might appear that this table has 6 rows and 8 columns, the internal table is not stored this way. There are two observations, one for Sex='F' and one for Sex='M'. Therefore, the output from Program 5.9 is actually just the block for Sex='F'. Furthermore, the variable that is listed here is not stored in a row of its own, but rather it's stored as a column. Therefore, there is no way to use the _MIN_, _MAX_, or _OBS_ keywords to subset the table so that it gives information only about variable='Height'.

The moral of the story is simple: Use these variables only when you have created the table or feel very comfortable with the layout that SAS uses to store the tabular output. You need to be able to specify the appropriate row. For some applications, sending the data to a data set via the OUTPUT destination is the more natural approach. This is especially true if the table is not easily addressed through variables with common names, such as TYPE. The OUTPUT destination is covered in Chapter 8, "Exporting to Data Sets."

Example: Reordering BY-Grouped Output

In the previous examples, you learned to save SAS output in an ODS destination and replay it later. But most of these examples really involved actions that could be accomplished in one step without the DOCUMENT procedure. This next example is quite challenging to try without the Document Facility.

In order to sense the power that is inherent in saving output now and digesting and printing it later, consider the following, which is a simplified version of an article in the SAS Knowledge Base. Our version omits the COPY statement since it has not been covered yet.

In the example document that has been built so far, you have seen that SAS stores output in the ODS document with each procedure's output in its own folder. Sometimes it is desirable to list all output, grouped by your BY variables instead. The default order is by procedure, and then grouped by BY group. If there were no DOCUMENT procedure, in order to gather all output for the variable SEX='M', you would have to read through each procedure and copy and paste the results into your report. That's an error-prone process. Much more compact is Program 5.10, which prints all output for SEX='F' followed by all output for SEX='M'.

Program 5.10: Reordering Output across BY Groups

```
ods html file="program5_10.html";
proc document name=mylib.firstdoc;
  replay ^(where=(sex='F'));
  replay ^(where=(sex='M'));
run;
quit;
ods html close;
```

In order to replay the top level directory with a WHERE statement, you must list the directory explicitly. The caret symbol (^) denotes the current directory, which is the root directory when the DOCUMENT procedure starts.

Try doing that without the DOCUMENT procedure!

Replaying Output with Temporary Formats

As long as templates and styles have not been altered, the replayed output will look exactly like the output that was produced from the original code. However, if the output needs specific formats to be displayed correctly, some care is necessary. Consider the following example, which saves and replays a document with a temporary format.

When temporary formats are used to create output, they are saved along with the data into the document itself. However, they are saved *invisibly*, and the user does not have access to them explicitly in the document tree.

Program 5.11 demonstrates how formats are managed during document replay.

Program 5.11: Saving a Document with a *Temporary Format* work.dummy

```
proc format;
  value $dummy 'F='1'
    'M='0';
run;

ods document name=mylib.smaller(write);

proc sql;
  select sex format=$dummy.,name,age,height
    from sashelp.class
    order by name;
quit;

ods document close;
```

Output 5.9: A Partial Listing

| The SAS System | | | |
|----------------|---------|-----|--------|
| Sex | Name | Age | Height |
| 0 | Alfred | 14 | 69 |
| 1 | Alice | 13 | 56.5 |
| 1 | Barbara | 13 | 65.3 |
| 1 | Carol | 14 | 62.8 |
| 0 | Henry | 14 | 63.5 |
| 0 | James | 12 | 57.3 |

For this example, Write mode is used to ensure that there is only one object in the document.

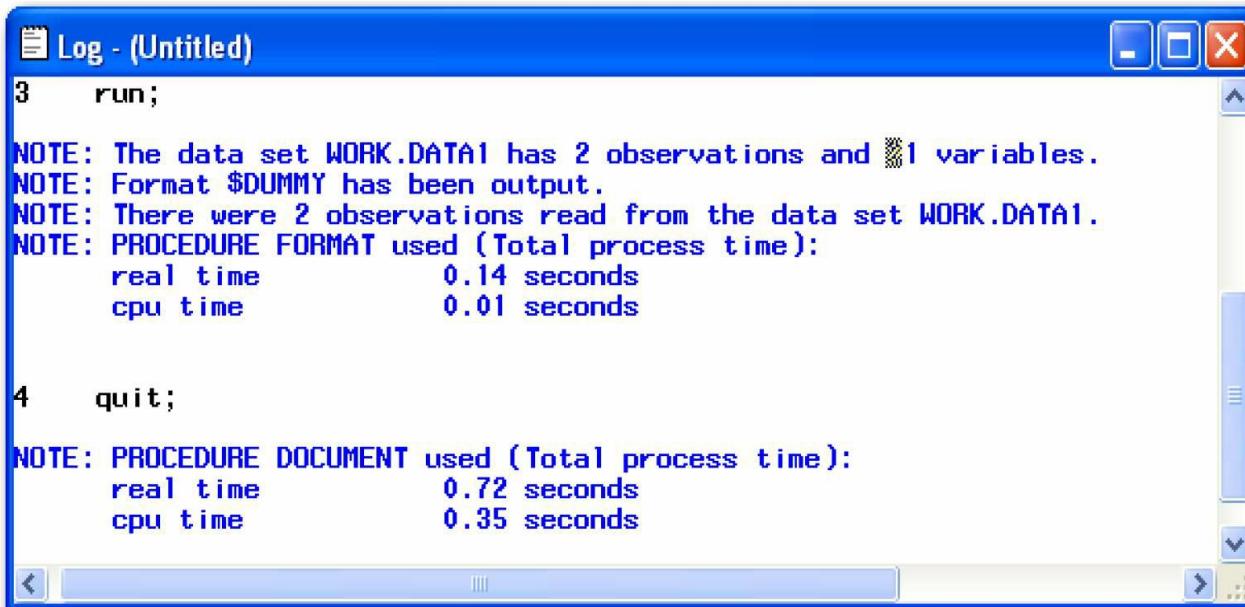
Close SAS and reopen it so that the temporary format `$dummy.` is no longer defined. Then run the replay code in Program 5.12. Alternatively, if you know how, just delete the temporary format from the format catalog.

Program 5.12: Replaying a Document with a Temporary Format

```
proc document name=mylib.smaller;
replay;
run;
quit;
```

When the document `mylib.smaller` is replayed, the message that is shown in Display 5.3 appears in the log file. Note the use of Write access mode. This is good for small examples if you want to ensure that the only output in the document is the output generated by the code you just ran.

Display 5.3: Log Showing Format Being Decompressed from the Document



The screenshot shows a window titled "Log - (Untitled)". The content of the log is as follows:

```
3 run;
NOTE: The data set WORK.DATA1 has 2 observations and 11 variables.
NOTE: Format $DUMMY has been output.
NOTE: There were 2 observations read from the data set WORK.DATA1.
NOTE: PROCEDURE FORMAT used (Total process time):
      real time      0.14 seconds
      cpu time      0.01 seconds

4 quit;
NOTE: PROCEDURE DOCUMENT used (Total process time):
      real time      0.72 seconds
      cpu time      0.35 seconds
```

If there were already a format named `$dummy` in the current WORK library, the REPLAY statement would override it. Thus, you must be careful when managing these *invisible* formats. There is no way to see these formats *as they are stored inside the document* either with the LIST statement or by other methods. They become visible only upon replay.

Temporary or Permanent Formats?

Although the ODS document persists temporary formats, you should consider carefully whether you want to use temporary or permanent formats in work that you save to an ODS document.

The advantage of temporary formats is that they are stored in the ODS document for you. If somebody else assumes responsibility for your document, they don't have to go searching for the formats needed. The correct temporary formats will be rebuilt in the current SAS session and will be used in all the replays.

By contrast, permanent formats are not stored in the document. Therefore, to produce output correctly when using the REPLAY statement, you must communicate where any user-defined formats are stored outside the document. However, this is not difficult. SAS has excellent methods of communicating about formats, including the format catalog. The FORMAT procedure offers the FMTLIB statement to easily display the meaning of each format. Formats can also be shared through CNTLOUT data sets. See the SAS documentation for the FORMAT procedure as well as the CATALOG procedure for listing the formats in a formats catalog.

Program 5.12 showed how the document implicitly stores temporary formats, and redefines them when the table is replayed.

Example with Permanent Formats

Program 5.13 shows the same program as 5.12, using permanent formats. SAS formats are a separate topic on their own, about which much has been written. The SAS Help and Documentation and the *SAS Language Reference: Concepts* are a good place to start if you are unfamiliar with the basics of permanent formats.

Program 5.13 shows how to change a format to change the appearance of a replay. When you store your work in a permanent format, changing the value of this format changes the appearance of the second replay. This example also features the use of WHERE=(_OBS_ ..) to keep the output short for the sake of making the illustration fit on the page.

Program 5.13: Using Permanent Formats

```
/* Let SAS know that you want to search for formats in MYLIB
   before WORK. This gives the permanent format precedence */
options fmtsearch=(mylib library work);
ods html style=highcontrast;
proc format lib=mylib; /* specify LIB= to make format permanent */
  value $dummy 'F'='1'
    'M'='0';
run;
/* same as program 5.8 */
ods document name=mylib.smaller(write);

proc sql;
  select sex format=$dummy3.,name,age,height
    from sashelp.class
    order by name;
quit;

ods document close;
/* first replay */
title 'With original value of MYLIB.DUMMY format';
proc document name=mylib.smaller;
  replay sql#1$sql_results(where=(_obs_ < 3)) / activetitle;
run;
quit;
```

```

/* reformat */
proc format lib=mylib;
value $dummy 'F'='1:F' 'M'='0:M';
run;

/* second replay */
title 'With new format';
proc document name=mylib.smaller;
replay sql#1$sql_results(where=(_obs_ < 3))/ activetitle;
run;

quit;

```

Output 5.10: Changing Replay by Changing a Format

With original value of MYLIB.DUMMY format

| Sex | Name | Age | Height |
|-----|--------|-----|--------|
| 0 | Alfred | 14 | 69 |
| 1 | Alice | 13 | 56.5 |

With new format

| Sex | Name | Age | Height |
|-----|--------|-----|--------|
| 0:M | Alfred | 14 | 69 |
| 1:F | Alice | 13 | 56.5 |

The advantage of the permanent format method is that you can change the definition of the permanent format at replay time. This principle is shown in the simple example of changing the format \$DUMMY, now stored in MYLIB. You can also change the definition of temporary formats as well, but if you close a session and start another session at a later time, replays continue to use the version of the format that was stored in the document.

Summary

The REPLAY statement replays a portion of your output, or it can replay everything that you have stored to date. The REPLAY statement by itself prints out the entire collected output.

As with the LIST statement, output that is generated by the REPLAY statements might be conditionally subset using the WHERE= option. The WHERE= option uses special variables, as discussed in Chapter 4, “Listing Documents Using the DOCUMENT Procedure.” New to this chapter were _MIN_, _MAX_, _OBS_ and the *observation variable name*, which enable you to replay any subset of rows of an output object. This feature is new for SAS 9.3.

When replaying documents, SAS re-installs any temporary formats that were in effect when the object was saved to the ODS DOCUMENT destination. Permanent formats are not saved into a document.

The same output can be printed in several formats, such as PDF, HTML, or LISTING, and more, all without rerunning the original analysis.

The examples shown also demonstrate that you can rearrange the order that SAS normally uses to present results; in this case, results are printed in BY-group order, even across procedures. Try doing *that* without the DOCUMENT procedure!

Chapter 6: Managing Folders

[Introduction](#)

[How ODS Document Folders are Different](#)

[Absolute Pathnames](#)

[Relative Pathnames](#)

[Positioning Arguments](#)

[When the TO Argument Requires a Sequence Number](#)

[WHERE= Clauses and Directories](#)

[The MAKE Statement](#)

[The DIR Statement](#)

[The RENAME TO Statement](#)

[Renaming a Document](#)

[Warning about Renaming Output Objects](#)

[The COPY TO Statement](#)

[A Common Mistake](#)

[Copying Using the LEVELS= option](#)

[The MOVE TO Statement](#)

[Browsing Collections of Documents with the DOC Statement](#)

[The LIBRARY= Option](#)

[The NAME= Option](#)

[The DELETE Statement](#)

[Warning: You Cannot Delete the Current Directory](#)

[The HIDE Statement](#)

[Extended Examples](#)

[Replay with the LEVELS= Option](#)

[Listing Every Document in Your System](#)

[Description of %alldocs Macro Arguments](#)

[Managing ODS Documents via the Graphical User Interface.](#)

[Summary](#)

Introduction

This chapter outlines all of the folder operations that the DOCUMENT procedure provides. Restructuring your document can make replay easier. For example, you could put all related output into a single folder so that replaying the entire unit for a report is quick and easy.

In this chapter, you will see the operations that are typical of a chapter about navigating file systems. There are statements you can use to copy, move, delete, and rename both folders and objects. One important difference is that the order of objects matters because the DOCUMENT procedure must replay a folder in order. Related to replay order, the HIDE and UNHIDE statements are also covered. These statements keep a section of your document from being replayed.

In addition to managing folders within a document, you also need some statements for managing your document collection as a whole. For this, there is the DOC statement. The DOC statement might look like an abbreviation of the ODS DOCUMENT statement from Chapter 3, “The ODS DOCUMENT Statement,” but do not confuse the two. They are completely different. The DOC statement is part of the DOCUMENT procedure.

The chapter concludes with two extended examples applying all of the techniques that have been discussed in this and the previous chapters.

How ODS Document Folders Are Different

As you have seen in Chapter 3, “The ODS DOCUMENT Statement,” SAS output is arranged into folders and subfolders based on the name of the procedure. It is possible to create your own folders. The statements that are used for manipulating folders are completely analogous to similar file navigation statements that you have learned elsewhere. However, there are a couple of important differences as they pertain to the document.

Absolute Pathnames

The syntax for an absolute pathname is as follows:

```
\<libname.>memname\path
```

For example, `\MYLIB.FIRSTDOC\univariate#1\` specifies the absolute pathname of the path `\univariate#1` in the `MYLIB.FIRSTDOC` document. It is necessary to have absolute pathnames when working with statements that handle paths that lie in a document that is not the current document. If both the source and destination paths are in the current document, it is not necessary to use the absolute pathname. Absolute pathnames are also covered in Chapter 8, “Exporting to Data Sets,” when discussing the SASEDOC engine.

Relative Pathnames

To reduce the amount of typing, many examples use relative pathnames. The way the DOCUMENT procedure uses pathnames is consistent with how folders work in most other computing environments. A relative pathname specifies a path based on your current location. Relative pathnames are distinguished by *not* being prefixed with a backslash, whereas absolute pathnames are *always* prefixed with a backslash.

A relative pathname takes one of the following two forms:

```
<folder1>\<folder2>\...<folder-m>
<folder1>\<folder2>\...><folder-m> \<object>
```

In order to find and replay an object, the DOCUMENT procedure concatenates the pathname of your current location along with the relative pathname.

Examples of relative pathnames are given throughout this chapter, and absolute pathnames are used only when *absolutely* necessary.

Positioning Arguments

The statements to move document folders and objects require not only a destination path, but also a position within the path. With the DOCUMENT procedure, you are arranging output *for replay*, and replaying requires a specific order. You must keep your desired replay order in mind as you manage your output. Therefore, you must specify the order, or accept the default. You do that by inserting at the end of the destination folder the results of the statements covered in this chapter.

The following statements require a positioning argument. Statements in italics are covered in later chapters.

```
COPY TO  
MOVE TO  
LINK TO  
IMPORT TO
```

The following positioning arguments are available:

```
BEFORE=path  
AFTER=path  
FIRST  
LAST
```

There might be more than one way to specify a position. If the folder is empty, FIRST and LAST are the same. Similarly, if you want to put an object in the third position, you can put it after the second or before the third.

The section about file operations provides examples of these positioning arguments. Chapter 8, “Exporting to Data Sets,” and Chapter 10, “Working with Links,” provide additional examples.

When the TO Argument Requires a Sequence Number

The TO argument might require a sequence number if you want to place objects into an existing folder. Otherwise, the DOCUMENT procedure creates a new folder with a new sequence number. This is an exception to the rule that omitting a sequence number defaults to the highest available sequence number. Examples of this are provided in the programs in the next section.

With the preliminary concepts established, it is time to begin using the statements for managing your output.

WHERE= Clauses and Directories

The statements in this chapter also support WHERE= clauses, but take note of the following warning from the *SAS Output Delivery System: User's Guide*:

For the REPLAY statement, the WHERE= option applies to directories and output objects. For the following statements, the WHERE= option applies to *directories* only:

COPY TO

DELETE

LIST

MOVE TO

In regard to WHERE= clauses on directory management commands: the target pathname in these commands is the pathname that is supplied. This remains true even if the WHERE= clause seems to suggest that only subdirectories would be affected by the command. You will see examples given in Programs 6.26 and 6.27.

The MAKE Statement

The MAKE statement creates a new, empty folder at a specific position in the supplied path.

Syntax of the MAKE Statement

```
MAKE path <, path-2, ...path-n> </ position-argument>;
```

Some applications of the MAKE statement include creating a document that contains all output of a specific type. For example, a document might contain all graphs, all TABULATE procedure output, all output from 1996, and all exhibits related to the final output of a project. In the examples that follow, you will make some empty folders, and you will put content in them using some of the other statements in this chapter.

One important advantage to using folders, instead of putting everything in the top-level or *root* folder, is that it is easier to manipulate related output as a single unit.

The simplest use of the MAKE statement is creating a folder at the end of a current path. This is the behavior when no options are provided. In Program 6.1 a folder named `graphs_only` is placed at the end of the current path, which is set to the root folder when the DOCUMENT procedure is invoked.

Program 6.1: Making a Subfolder at the End of the Current Path

```
proc document name=mylib.quickstart;
  make graphs_only;
/* You could also have written the MAKE statement */
/* using the equivalent forms */
* make \graphs_only;
* make \mylib\firstdoc\graphs_only;
  title 'New Folders at End';
  list;
  run;
  quit;
```

If you prefer your output to be at the beginning of the current path, use the / FIRST option.

Program 6.2 shows the / FIRST option used to create a folder at the beginning of the current path.

The equivalent forms that are mentioned use the full pathname and the absolute pathname, respectively. These forms are longer than are necessary to specify the same directory. However, they can appear in macro programs, or in other applications that generate code automatically, such as Program 6.35 at the end of this chapter.

Program 6.2: Making Subfolders at the Beginning of the Current Path

```
proc document name=mylib.quickstart;
  make us_only / first;
  make europe_only / after= us_only;
  make asia_only / after= europe_only;
  title 'New Folders at Front';
  list;
  run;
  quit;
```

You are not limited to making folders in the top level. You can also make subfolders of any folder as shown in Program 6.3.

At the beginning of a new project, a common task is to examine a data set and assess its

suitability for your purpose. The reports that you run might be voluminous; you might need to save them and answer questions in a hurry if somebody asks. If they do, you might not have the time to rerun the analysis that created the reports. The DOCUMENT procedure is perfect for such a task. Program 6.3 shows one such folder. You might have other ways that you envision structuring your folders to best access your output. The backslash in the code is a line continuation. It is not intended to be part of the label.

Program 6.3: Making a Subfolder in the Path Below

```
proc document name=new(write);
  make dataquality;
  setlabel dataquality 'Basic reports to assess a new data set \
such as Freqs, Missing Values, Means';
  make dataquality\histograms;
  make dataquality\character_vars;
  list / details levels=3;
  run;
quit;
```

The resulting document is shown in Output 6.1. In order to fit the output to the page, some columns were omitted. If you would like to see how this was done, see Chapter 11, “Working with Templates.”

Output 6.1: An Empty Document with More Than One Level of Folders

| New Directories | | |
|---------------------------------|------|---|
| Listing of: \Work.New\ | | |
| Order by: Insertion | | |
| Number of levels: 3 | | |
| Path | Type | Label |
| \dataquality#1 | Dir | Basic reports to assess a new data set such as Freqs, Missing Values, Means |
| \dataquality#1\histograms#1 | Dir | |
| \dataquality#1\character_vars#1 | Dir | |

With the folders in place, you might be wondering how to store output in your new folders. There are several ways. After sending code to the ODS DOCUMENT destination, you can use the MOVE TO statement to customize the order of your output. This is demonstrated in Program 6.13 through use of the COPY TO and MOVE TO statements.

It is helpful in some applications to set up your folder structure before starting work. For example, a pharmaceutical company might organize product development work into folders as shown in Program 6.4.

Program 6.4: Setting Up a Document Before Starting Work

```
proc document name=my.lib.drugs;
  make new_cardiac_drug;
  make cardiac_drug\marketing_surveys;
  make cardiac_drug\clinical_trials;
  make cardiac_drug\clinical_trials\pt_recruitment;
  make new_cholesterol_drug;
  /* etc */
  run;
```

When it is time to populate the document with procedure output, use the DIR= option in the ODS

DOCUMENT statement, presented here as a review.

Program 6.5: Using the DIR= Option to Populate Folders

```
ods document name=mylib.drugs dir=(\new_cardiac_drug\marketing_survey);
/* sas code */
ods document close;
```

The DIR Statement

The DIR statement changes the current folder. When the DOCUMENT procedure is invoked, the default folder is the root folder. Document management becomes much easier when items are stored in folders, as compared with storing everything at the root. This is because it is easier to move folders around, but the root, by definition, cannot be moved.

Syntax of the DIR Statement

```
DIR <pathname>;
```

If the pathname is omitted, the DIR statement prints the *absolute pathname* of the current path to the open ODS destination.

The DIR statement is more like a *change directory* rather than like DIR in DOS environments. Remember that the LIST statement fulfills the purpose of listing them. Program 6.6 demonstrates examples of folder navigation using relative pathnames. Remember that two carets (^) denote the parent folder. So that you can see the effect of each DIR statement, each statement is followed by a DIR statement with no pathname. Normally, you would not need to print the current folder as often as Program 6.6 does.

Program 6.6: Navigating Folders

```
ods html style=htmlblue;
ods noproctitle;
options nobyline;
proc document name=mylib.quickstart;

dir univariate;
dir by group1#1; /* Origin=Asia, type=Hybrid */
dir;
title "Asian Car Wheelbase Data (in inches)";
replay wheelbase\basicmeasures;
run;

title;
dir ^^; /* navigate up to parent folder */
dir; /* show current folder */
run;

title "For SUVs, now...";
dir by group2#1; /* Origin=Asia, type=SUV */
dir; /* show folder again */
replay wheelbase\basicmeasures;
run;

/* in one statement, navigate to by group3#1; */
dir ^^\by group3;
run;

quit;
ods html close;
```

Relevant sections of the output are shown in Output 6.2, which has been edited for space. Notice how the folder appears above the item to be printed. This can make the object easy to find at a later time.

Output 6.2: Showing Results of Navigation Statements

Asian Car Wheelbase Data (in inches)

\Mylib.Quickstart\Univariate#1\ByGroup1#1

The SAS System

| Basic Statistical Measures | | | |
|----------------------------|----------|---------------------|----------|
| Location | | Variability | |
| Mean | 101.3333 | Std Deviation | 5.68624 |
| Median | 103.0000 | Variance | 32.33333 |
| Mode | . | Range | 11.00000 |
| | | Interquartile Range | 11.00000 |

\Mylib.Quickstart\Univariate#1

For SUVs, now...

\Mylib.Quickstart\Univariate#1\ByGroup2#1

The SAS System

| Basic Statistical Measures | | | |
|----------------------------|----------|---------------|---------|
| Location | | Variability | |
| Mean | 108.0400 | Std Deviation | 7.06800 |

You can now create your own folders. You might want to rename them, or even rename the folders that SAS creates automatically.

The RENAME TO Statement

The RENAME TO statement renames any folder or output object. You can also rename the document itself.

Syntax of the RENAME TO Statement

```
RENAME path-1 TO path-2
```

Renaming a Document

The first example of the RENAME statement, Program 6.7, shows how to rename an entire document. This program runs even if you do not have a document named MYLIB.BIGGER. Recall from Chapter 4, “Listing Documents Using the DOCUMENT Procedure,” that the PROC DOCUMENT statement creates a new document if one does not exist.

Program 6.7: Renaming a Document

```
proc document name=mylib.Bigger;
rename Mylib.bigger TO Mylib.atomic;
run;
```

A common use of the RENAME statement arises when you might have multiple sequence numbers of the same folder, but you want to remember them by different names.

As a project gets larger, you will have more need for renaming folders. Within a document, you can re-label folders, as Program 6.8 demonstrates.

Program 6.8: Renaming a Folder

```
ods html file="program6_8.html" newfile=none;
proc document name=show(write);
make folder;
make folder\subfolder;
make folder\subfolder; /* will make subfolder#2 */
/* move into folder */
dir folder;
setlabel subfolder#1 "Will you remember what 'subfolder#1' contains?";

dir ^^;
title "Before";
list / details levels=2;
run;

proc document name=show(update);
title "After Rename";
rename folder#1 to retail_report ;
setlabel retail_report 'Logistics Reports by Region, 2010-2013';

dir retail_report;
rename subfolder#1 to east_region;
setlabel east_region 'East Region Reports';

rename subfolder#2 to west_region;
setlabel west_region 'West Region Reports';
dir ^^; /* go to parent directory */

list / details levels=2;
run;
quit;
ods html close;
```

In this small experimental document, work.show, a folder and two subfolders are created. Output 6.3 shows the output from the two LIST statements before and after the RENAME statement.

Notice that the second MAKE statement is the same as the first. This creates a folder with the same name and a new sequence number. Although it is unlikely that you would deliberately create two different folders with the same name, this situation could arise. For example, you might want to re-run the same code several times with minor modifications each time.

Output 6.3: Output Before and After the RENAME Statement

| Before | | | | | | | | |
|------------------------|------|------------------|--------------------|--------------------|---------------|----------|--|------------|
| Listing of: \Work.Show | | | | | | | | |
| Order by: Insertion | | | | | | | | |
| Number of levels: 2 | | | | | | | | |
| Path | Type | Size in Bytes | Created | Modified | Symbolic Link | Template | Label | Page Break |
| \Folder#1 | Dir | | 19JUN2012:11:58:54 | 19JUN2012:11:58:54 | | | | |
| \Folder#1\Subfolder#1 | Dir | | 19JUN2012:11:58:54 | 19JUN2012:11:58:54 | | | Will you remember what 'Subfolder#1' contains? | |
| \Folder#1\Subfolder#2 | Dir | | 19JUN2012:11:58:54 | 19JUN2012:11:58:54 | | | | |

| After Rename | | | | | | | | |
|--------------------------------|------|------------------|--------------------|--------------------|---------------|----------|--|------------|
| Listing of: \Work.Show | | | | | | | | |
| Order by: Insertion | | | | | | | | |
| Number of levels: 2 | | | | | | | | |
| Path | Type | Size in Bytes | Created | Modified | Symbolic Link | Template | Label | Page Break |
| \Retail_Report#1 | Dir | | 19JUN2012:11:58:54 | 19JUN2012:11:58:55 | | | Logistics Reports by Region, 2010-2013 | |
| \Retail_Report#1\East_Region#1 | Dir | | 19JUN2012:11:58:54 | 19JUN2012:11:58:54 | | | East Region Reports | |
| \Retail_Report#1\West_Region#1 | Dir | | 19JUN2012:11:58:54 | 19JUN2012:11:58:54 | | | West Region Reports | |

Program 6.9 completes the process of making your document listing more reader-friendly. The pathname is related to the contents and a descriptive label fills in the details.

Program 6.9: Making Your Document Listing Reader-Friendly

```
proc document name=show(update);
  rename folder to Cardiac_Drugs;
  dir Cardiac_Drugs;
  rename subfolder#1 to sales_2010;
  rename subfolder#2 to sales_2011;
  setlabel sales_2010 'Ah, Much better';
  DIR ^^;
run;
title "After";
list / details levels=all;
run;
quit;
ods html close;
```

When the folder is renamed, the sequence numbering is restarted at #1. It is redundant to put a sequence number in the TO position of a RENAME statement. The log excerpt in Output 6.4 shows the error message when the marked line is changed to read "..TO sales_2010#1".

Output 6.4: Log Error Message

```
675      rename folder to Cardiac_Drugs;
676      dir Cardiac_Drugs;
677
678      rename subfolder#1 to sales_2010#1;
-
22
200
NOTE: The previous statement has been deleted.
ERROR 22-322: Expecting ;
ERROR 200-322: The symbol is not recognized and will be ignored.
```

Warning about Renaming Output Objects

After renaming documents and then folders, the next logical step in detail would seem to be renaming output objects. Although this is technically permissible, for the purposes of this chapter, it is better to leave output objects alone. When working with the output of SAS procedures, renaming breaks the association between the standard names for those tables and your work. For example, if you rename the `ParameterEstimates#1` object to something shorter, such as `Params#1`, someone who views your document might not realize that your object `Params1#1` is actually the `ParameterEstimates#1` output from the REG procedure. Deprived of this connection, they might not be able to discern the origin of your output or fully make use of the standard names of those tables.

The most important reason not to rename output objects is that the standard names are well documented, whereas somebody reading your code would have to work harder to discern the purpose of your renamed table. Therefore, it is recommended that you limit yourself to renaming only documents and folders that you create, leaving folders created by SAS alone.

The COPY TO Statement

The MAKE and DIR statements are a step toward custom file organization, and they work like their analogs in other file systems. The COPY TO statement is the first of several that require positioning arguments. The syntax of the copy statement is shown next.

Syntax of the COPY TO Statement

```
COPY path <(where-expression)> <, path-2<(where-expression-2)>>
<, ...path-n<(where-expression-n)>> TO path </option(s)>;
```

There is another principle that helps you remember the difference between the roles of the left and right side of the TO keyword.

If the PROC DOCUMENT statement accepts multiple comma-separated entries, then one or more of the entries can be an *item store library.member* name. This is true for COPY and MOVE on the left side of the TO. It is also true for the LINK TO statement discussed in Chapter 10, “Working with Links.” Program 6.10 demonstrates the critically important task of backing up a document.

Program 6.10: Backing Up a Document

```
proc document name=my.lib.backup(write);
  copy my.lib.firstdoc to ^;
  run;
quit;
```

The DOCUMENT procedure needs a current document, so it creates an empty document to receive the contents of MYLIB.FIRSTDOC. The COPY TO statement copies the entire document to the current folder. If you want to copy everything to a subfolder instead of the root folder, use Program 6.11, which combines the MAKE, DIR, and COPY TO statements.

Program 6.11: Backing Up a Document to a Subfolder

```
ods html file="program6_11.html";
proc document name=try(write) label="WORK.TRY: Backup of MYLIB.FIRSTDOC";
  make old_stuff;
  setlabel old_stuff 'Original Document used to write book';
  dir old_stuff;
  copy my.lib.firstdoc to ^;
run;
list / levels=2 details;
quit;
ods html close;
```

Output 6.5 shows the new document, with the folder structure and contents from MYLIB.FIRSTDOC copied intact.

Output 6.5: Backup of MYLIB.FIRSTDOC

Listing of: \Work.Try\old_stuff#1

Order by: Insertion

Number of levels: 2

| Obs | Path | Type |
|-----|--------------------------------------|------|
| 1 | \old_stuff#1\Univariate#1 | Dir |
| 2 | \old_stuff#1\Univariate#1\ByGroup1#1 | Dir |
| 3 | \old_stuff#1\Univariate#1\ByGroup2#1 | Dir |
| 4 | \old_stuff#1\Freq#1 | Dir |
| 5 | \old_stuff#1\Freq#1\Table1#1 | Dir |
| 6 | \old_stuff#1\Reg#1 | Dir |
| 7 | \old_stuff#1\Reg#1\ByGroup1#1 | Dir |
| 8 | \old_stuff#1\Reg#1\ByGroup2#1 | Dir |
| 9 | \old_stuff#1\Reg#2 | Dir |
| 10 | \old_stuff#1\Reg#2\ByGroup1#1 | Dir |
| 11 | \old_stuff#1\Reg#2\ByGroup2#1 | Dir |

You could store several backups in this way, using one folder for each. This would reduce the number of separate documents you need to manage, because you would be using one folder per backup instead of one document.

Program 6.12 shows another method to send a folder in one document to another. Instead of creating a blank document as the current document, this uses the source document as the current one, and then sends it outbound to the destination. This is called the *outbound* style of copying, as contrasted with the *inbound* style of the previous example. The destination, since it is outside the current document, must be an absolute pathname.

Program 6.12: Sending a Folder from One Document to Another

```
proc document name=my.lib.again(write) label='Copy Examples';
quit;

proc document name=my.lib.firstdoc;
copy ^ to \my.lib.again\;
run;
quit;
```

In Program 6.12 the first call to the DOCUMENT procedure shows one of many ways to create a new, empty document. Then, the main document step in this example establishes the document my.lib.firstdoc as the current document, and sends the current folder as a subfolder of the document my.lib.again.

If you plan to work with two or more documents, and need to frequently switch which one is current, you might find it easier to use the DOC statement, and make only one call to the DOCUMENT procedure. This is illustrated in the section about the DOC statement, later in this chapter.

A Common Mistake

The COPY TO statement permits WHERE= option statements, just as REPLAY does. They work exactly the same here; all variables including underscore variables and BY-group variables, if applicable, can be used in WHERE= statements.

If you are copying within the same document, be careful not to make the source an ancestor of the destination folder. Although it is unlikely that you would do this deliberately, it is easy to make this type of mistake when working with the top-level folder. Program 6.13 demonstrates this error.

Program 6.13: Attempting to Copy a Folder to a Subfolder of Itself

```
proc document name=mylib.firstdoc;
  make graphs_only;
/* problem */
  copy ^(where=(_type_ = 'Graph')) to graphs_only;
/* OK - source path (univariate) is not an ancestor to
destination(graphs_only)*/
  copy univariate#1(where=(_type_= 'Graph')) to graphs_only#1;
quit;
```

Although it might seem that we are transferring only files, not folders, SAS produces an error message:

Display 6.1: Log Error Message

ERROR: Source path is ancestor of destination path.

This error is occurring because the ^ folder is the ancestor of the folder `make_graph`.

Sometimes you might want to copy output from a document into a new folder in that same document. But doing so would violate the rule about the source path being the ancestor of the destination path. How can this issue be worked around? One preventative measure is to make it a habit to store output in a subfolder from the outset, as illustrated in Program 6.14.

Using the example from Chapter 3, if Program 3.1 had been written as follows, then it would be easy to move information from the folder `chapter3#1` to the folder `graphs_only#1` because the source path, `\chapter3` is no longer the destination of the ancestor path, “`graphics_dir`”.

Program 6.14: Storing Output In a Subfolder

```
ods document name=mylib.betterdoc path=(dir=\chapter3 label='Redone');
/* code from program 3.1 */
ods document close;
```

Another advantage to storing each run of your code in a separate folder is that it becomes easier to manage because you could then give each folder its own descriptive label. The folders can then be copied, moved, deleted, or generally managed as a distinct unit without inadvertently affecting your other analyses.

Another possibility is to create the target folder in another document, as demonstrated in Program 6.15. Then, there would be no conflict between the source and destination folders.

Program 6.15: Creating the Target Folder In Another Document

```
proc document name=mylib.allgraphs(write)
  label='Graphics Objects from Mylib.firstdoc';
  make graphics_dir;
run;
```

```
copy \mylib.firstdoc\^(where=(_type_='Graph')) to graphics_dir#1;
run;
quit;
```

If needed, you could follow this up by copying the `graphics_dir` folder back to the original document, or you could decide to maintain the new folder separately as part of the new document. Program 6.16 demonstrates the copy to the root level directory of document `mylib.firstdoc`.

Program 6.16: Copying to the Root Level Directory

```
proc document name=mylib.allgraphs(read);
  copy graphics_dir to \mylib.firstdoc\ / levels=all;
  run;
quit;
```

The COPY TO statement also accepts WHERE= options on all source pathnames that it takes as arguments. Source pathnames are those that come before the TO statement. Program 6.17 creates a document that contains only graphs. Files that contain only output objects (and this includes links as discussed in Chapter 10, “Working with Links”) can be useful in creating custom reports.

Program 6.17: Using the WHERE= Option to Selectively Copy Based on Conditions

```
title 'Backup using a WHERE statement';
proc document name=sasuser.archive(write);
list \mylib.firstdoc\ details;
run;

make january;
make february;
make march;
copy \mylib.firstdoc\^(where=(_cdate_>'01Mar2012'd)) to march#1;
run;
list / levels=2;
run;
quit;
```

Output 6.6 shows the output. Notice that this copy occurs between two different documents.

Output 6.6: Conditionally Copying Based on Date

| Backup using a WHERE statement | | |
|--------------------------------|-----------------------|------|
| Listing of: \Sasuser.Archive\ | | |
| Order by: Insertion | | |
| Number of levels: 2 | | |
| Obs | Path | Type |
| 1 | \january#1 | Dir |
| 2 | \february#1 | Dir |
| 3 | \march#1 | Dir |
| 4 | \march#1\Univariate#1 | Dir |
| 5 | \march#1\Freq#1 | Dir |
| 6 | \march#1\Reg#1 | Dir |
| 7 | \march#1\Reg#2 | Dir |

You can also copy from the current document to an external document. Program 6.18 shows this version of the COPY statement.

Program 6.18: Destination of COPY Is External to Current Document

```

title 'Outbound Style';
proc document name=my.lib.firstdoc;
  copy ^(where=(_cdate_>'01Mar2012'd)) to \sasuser.archive\march#1;
  run;
quit;

```

You can copy within documents as well as between them. Program 6.19 creates a copy of all the table objects pertaining to the UNIVARIATE procedure. The remainder of the examples in this section focus on copying a folder in a document to another folder in the current document.

Program 6.19: Copy Folder to Current Path

```

ods html file='program6_19.html';
proc document name=my.lib.firstdoc;
  copy univariate#1 to ^ / last;
  setlabel univariate#2 'Copy of Univariate Folder';
  list / details;
  run;
quit;
ods html close;

```

Output 6.7 shows the listing of the document, including the new folder. You might consider such a copy if you plan to make alterations to one version. However, the real purpose of this program is to understand how this copy works with a WHERE= clause.

Output 6.7: Creating a Copy of a Folder

| Listing of: \Mylib.Firstdoc | | | | | | | | | |
|-----------------------------|---------------|------|---------------------|--------------------|--------------------|------------------|----------|---------------------------|---------------|
| Order by: Insertion | | | | | | | | | |
| Number of levels: 1 | | | | | | | | | |
| Obs | Path | Type | Size in Bytes | Created | Modified | Symbolic Link | Template | Label | Page Break |
| 1 | \Univariate#1 | Dir | | 29MAR2012:21:59:04 | 29MAR2012:21:59:04 | | | The Univariate Procedure | |
| 2 | \Freq#1 | Dir | | 29MAR2012:21:59:04 | 29MAR2012:21:59:04 | | | The Freq Procedure | |
| 3 | \Reg#1 | Dir | | 29MAR2012:21:59:04 | 29MAR2012:21:59:04 | | | The Reg Procedure | |
| 4 | \Reg#2 | Dir | | 29MAR2012:21:59:04 | 29MAR2012:21:59:04 | | | The Reg Procedure | |
| 5 | \Univariate#2 | Dir | | 11APR2012:21:20:03 | 11APR2012:21:20:03 | | | Copy of Univariate Folder | |

The output to Program 6.20 might not be what you expect, but if you understand how Program 6.19 works, the explanation is more straightforward.

Program 6.20: Copy Output Objects in a Folder to Current Path

```

ods html file='program6_20.html';
proc document name=my.lib.firstdoc;
  copy univariate#1(where=(_type_= 'Table')) to ^ / levels=all;
  setlabel univariate#3 'Output Objects from Univariate Only.
  No Subfolders (seqno=#3)';
  run;
  list;
  run;
  quit;
ods html close;

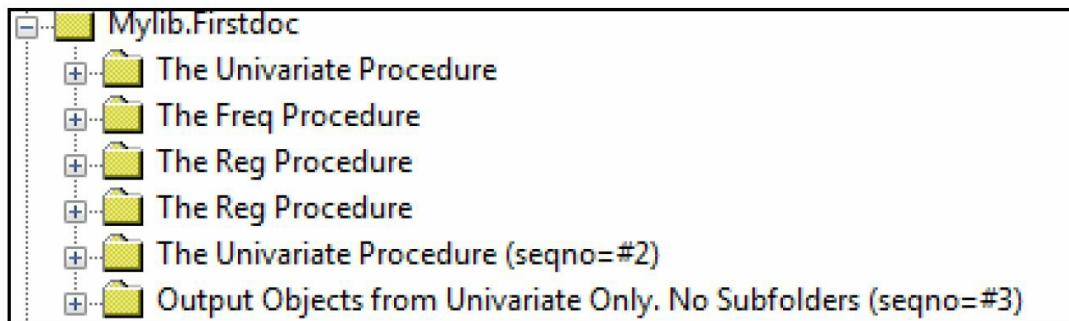
```

It is critically important to understand how this COPY statement works. It creates a version of the Univariate#1 folder in the root directory, calls it univariate#3, and puts the output objects in this directory. If you imagine that the output would be like Display 6.2, with the output objects placed directly at the ROOT, you would be mistaken. To see why, it is best to imagine the COPY TO statement without its WHERE clause.

```
COPY univariate#1 to ^ / first;
```

You have seen this before. This is precisely the COPY TO statement from Program 6.16. In that instance, it was not surprising that the directory structure looked like that of Display 6.2.

Display 6.2: How Conditional Copying Actually Works

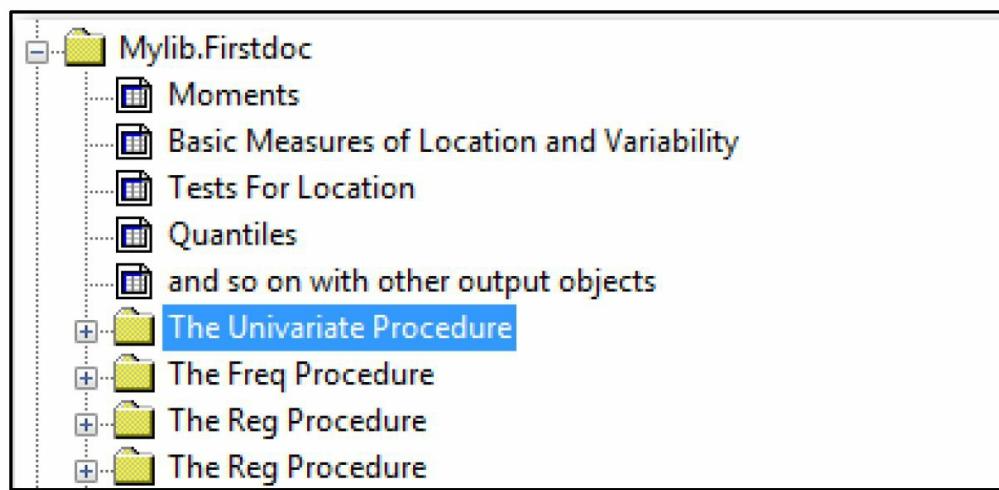


With this setup, it should be clearer that the univariate#1 folder is copied to the current directory. Since there are already folders named univariate#1 and univariate#2 in the current directory, the system must create a new folder with a higher sequence number. This would be univariate#3.

The document, after the COPY TO statement with the WHERE= clause, would be this same structure, but the folder univariate#3 would contain only those objects satisfying the WHERE= clause. The section of the listing relevant to this discussion is shown in Output 6.8. Univariate#3 has a flat structure, consisting of all the tables from the UNIVARIATE procedure. These types of folders have some advantages that will be explored further in Program 6.34 and again in the next chapter. It is not mandatory to use them.

Output 6.8: UNIVARIATE Procedure Output

| | | |
|----|----------------------------------|-------|
| 12 | \Univariate#2\ByGroup1#1 | Dir |
| 13 | \Univariate#2\ByGroup2#1 | Dir |
| 14 | \Univariate#3 | Dir |
| 15 | \Univariate#3\Moments#1 | Table |
| 16 | \Univariate#3\BasicMeasures#1 | Table |
| 17 | \Univariate#3\TestsForLocation#1 | Table |
| 18 | \Univariate#3\Quantiles#1 | Table |
| 19 | \Univariate#3\ExtremeObs#1 | Table |
| 20 | \Univariate#3\Moments#2 | Table |
| 21 | \Univariate#3\BasicMeasures#2 | Table |
| 22 | \Univariate#3\TestsForLocation#2 | Table |
| 23 | \Univariate#3\Quantiles#2 | Table |



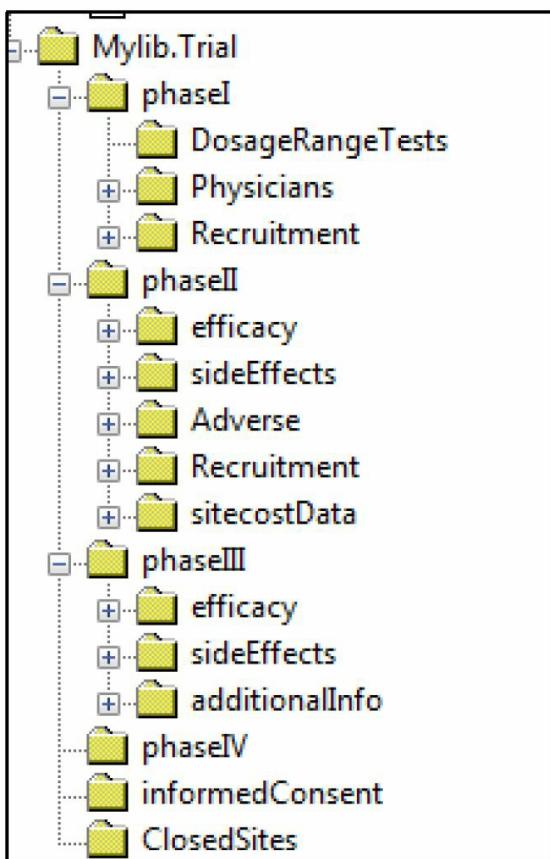
It is not advantageous to put too much in the root directory. The root directory cannot be moved or deleted. To manage objects that you want to treat as a group, place them together in their own folder.

Here's another way to think about this. When you use COPY TO, it's like you are taking a vacation with the folder acting like the suitcase. You will take this suitcase with you to the destination, even if you have restricted what goes in it, as with a WHERE clause. The COPY TO statement does not unpack for you when you arrive.

Copying Using the LEVELS= Option

Any DOCUMENT statement that can reasonably take a LEVELS= option will do so, for the sake of consistency. The statements that take a LEVELS= option include the COPY TO and MOVE TO statements discussed in this chapter, as well as the LIST and REPLAY statements discussed in previous chapters. As the final example of copying, you will copy the document's overall “big picture” folder structure, without any output objects. This approach can be useful when you have fairly standard projects, and only the details change from project to project. Mylib.trial represents a document for a fictional clinical trial. Assume that, over time, a large volume of SAS procedure output would be stored in each folder. (This is not intended to represent or recommend actual trial practice.)

Display 6.4: Copying Using the LEVELS= Option



Program 6.21 would copy just the first two levels of folders, leaving the output behind in the original document.

Program 6.21: Copying with the LEVELS= Option

```
proc document name=mylib.trial(write);  
make phaseI;  
make phaseI\DosageRangeTests;  
make phaseII;  
make PhaseII\efficacy;  
make PhaseII\sideEffects;  
make phaseII\sideEffects\scratchpad;  
make PhaseII\Adverse;  
make phaseIII;  
make phaseIII\efficacy;  
make phaseIII\sideEffects;  
make phaseIII\additionalInfo;  
make phaseIV;  
make phaseI\Physicians;  
make phaseI\Recruitment;  
make phaseII\Recruitment;  
make informedConsent;
```

```
make phaseII\sitecostData;  
make ClosedSites;  
run;  
quit;  
  
proc document name=newtrial;  
copy mylib.trial to ^ / levels=2;  
run;
```

The output shows that the Level 3 directory, `phaseII\sideEffects\scratchpad` is not included in the document `newtrial`.

The MOVE TO Statement

All of what was discussed regarding copying folders and objects applies equally well to the MOVE TO statement. Keep in mind, however, that the original objects have been moved instead of copied. The key points are that WHERE= clauses are allowed and that you cannot move a directory to a subdirectory of itself. The source cannot be the ancestor of the destination. That discussion can be assumed, without repeating the analogous examples.

The MOVE TO statement has another application, which is the focus of this section, and which typically does not occur in other file navigation systems. Namely, because order is significant in the DOCUMENT procedure, you can use MOVE TO statements to reorder objects within a single directory.

Using the WORK.SHOW document above, Program 6.22 adds two new folders. Because there is no positioning argument, they are added to the end of the folder. The MOVE TO statement brings the 2011 folder to the beginning of the `cardiac_drug` folder.

Program 6.22: Managing the Order of Files Using the MOVE TO Statement

```
proc document name=mylib.drugs;
  make cardiac_drug;
  dir cardiac_drug;
  make sales_2011,sales_2010,sales_2009,sales_2008;
  move sales_2011 to ^/first;
  list;
  run;
  quit;
```

Output 6.9 shows that the new folder has a new sequence number. Since there was a SALES_2011#1 in the file just before the statement is executed, the sequence number must be incremented.

Output 6.9: Document Recorded in Descending Order

```
Listing of: \Work>Show\Cardiac_Drugs#1
Order by: Insertion
Number of levels: 1
```

| Obs | Path | Type |
|-----|---------------------------|------|
| 1 | <code>sales_2011#2</code> | Dir |
| 2 | <code>sales_2010#1</code> | Dir |
| 3 | <code>sales_2009#1</code> | Dir |
| 4 | <code>sales_2008#1</code> | Dir |

Of course, you can move a folder to another location in the document. In Program 6.23 a new folder is created to hold the output that is stored in MYLIB.FIRSTDOC, and a new folder is created to store subsequent output. This is a good way to preserve older work while not overwriting it with newer work.

Program 6.23: Moving Output to a Subfolder

```
proc document name=mylib.dccopy(write);
  copy mylib.firstdoc to ^; ①
  make new_stuff;
  setlabel new_stuff 'Material for Second Edition';
  make old_stuff;
  setlabel old_stuff 'Material from First Edition';
```

```
move univariate#1,freq#1,reg#2 to old_stuff; ②  
run;  
quit;
```

- ❶ This line creates a new document, `mylib.dccopy`, or deletes all of its contents if it exists. It then copies the contents of `mylib.firstdoc` and places it in the current folder, which is the root. This is necessary only insofar as when you first start working with the MOVE TO statement, it's best to work with a copy of your document. Once you feel comfortable with what MOVE TO does, you can stop adding this security measure to the code.
- ❷ This MOVE TO statement moves the three folders into the folder `old_stuff#1`. Now they will stay out of the way of new work, but still be accessible. Note that this same file structure could have been accomplished by copying to the destination `new_stuff#1` in a single command. However, the purpose of the copy was simply to have some material in the document on which to demonstrate the MOVE TO statement.

Browsing Collections of Documents with the DOC Statement

The LIST statement gives detailed information about a single document, but sometimes you might need a list of all the documents that were created. The DOC statement is versatile. It can list all the documents on a system, or all the documents in a library. It can assign labels as well to documents. It can also change the current document name without closing the DOCUMENT procedure.

To reiterate though, the DOC statement is not to be confused with the ODS DOCUMENT statement from Chapter 3.

The Syntax for the DOC Command

```
DOC <options<access-option(s)>>;
```

You can use the following options.

NAME=;

LIBRARY=;

CLOSE;

All arguments are optional; however, LABEL makes sense only if the NAME= option is used. Access modes are exactly as described in Chapter 3, “The ODS DOCUMENT Statement.”

The simplest case is to submit the DOC statement without any arguments. Doing so lists all of the documents that are available in the current session.

Program 6.24: Shows a Listing of All the Documents Available in a Session

```
proc document  
  doc;  
  run;  
  quit;
```

The output is presented in Output 6.10. Your output might vary slightly if you have already created some documents of your own. Those documents will also appear in this listing.

Many of these documents do not have labels. As a result, it is difficult to know what these documents contain unless you run the LIST statement on each document.

Output 6.10: All Available Documents

All Documents

| Documents | | |
|-----------|------------------|------------------------------|
| Obs | Name | Label |
| 1 | Mylib.Backup | |
| 2 | Mylib.Firstdoc | |
| 3 | Mylib.Graph | |
| 4 | Mylib.Lastone | Analysis for 2004 Model Cars |
| 5 | Mylib.Quickstart | |
| 6 | Mylib.Seconddoc | |

Since the point of this book is to help you manage your output, the next step is to remedy this situation.

The LIBRARY= Option

The LIBRARY= option, which can also be aliased as LIB=, produces a list of documents in the specified library. Since its function is so similar to the DOC statement with no options, no additional examples are provided. One helpful use of this command is to list all documents in the WORK library. To avoid losing these when the session closes, you must copy these documents to a permanent item store.

Program 6.25: Listing Temporary Documents

```
ods html file='program6_25.html';
title 'Temporary';
proc document;
  doc lib=work;
  run;
quit;
ods html close;
```

Output 6.11 shows the temporary documents that were created for this chapter. You might have additional documents in your system.

Output 6.11: Listing of all Documents in the WORK Library

| Temporary | | |
|-----------|---------------|--|
| Documents | | |
| Obs | Name | Label |
| 1 | Work.Chap7 | |
| 2 | Work.Firstdoc | Temporary Copy of MYLIB.FIRSTDOC to demonstrate destructive operations |

The NAME= Option

The NAME= option opens a document in the appropriate access mode and makes that document current. Using DOC with the NAME= option is completely equivalent to executing the following statement:

```
proc document name=;
```

So, both NAME= options fulfill the same purpose. The difference in usage is that with the DOC statement, you can change the current documents without stopping the DOCUMENT procedure. *Besides this convenience feature, there is no difference in function.* Program 6.23 illustrates working with two different current documents during a single run of the DOCUMENT procedure. Even when you are handling multiple documents in a single DOCUMENT procedure call, only one document can be current at any one time. If any of these documents do not exist, the DOCUMENT procedure creates them. Program 6.26 provides another way of creating a new document, or several at once.

Program 6.26: DOC NAME= Opens a Document

```
ods html file='program6_26.html';
proc document;
  title "Quick Start Document from Chapter 1";
  doc name=mylib.quickstart label='Analysis of SASUSER.CARS';
  list;
  run;
  doc close;

  title "Firstdoc Document from Chapter 3";
  doc name=mylib.firstdoc;
  list;
  run;
  doc close;

  title "All Documents in all libraries, Showing Labels";
  doc ;
  run;
  quit;
  ods html close;
```

Output 6.12: Using Several Documents in a Single Call of the DOCUMENT Procedure

First Document: Quick Start Document from Chapter 1

| Listing of: \Mylib.Quickstart(where=(_seqno_=1)) | | |
|--|--------------|------|
| Order by: Insertion | | |
| Number of levels: 1 | | |
| Obs | Path | Type |
| 1 | Contents#1 | Dir |
| 2 | Univariate#1 | Dir |
| 3 | Means#1 | Dir |

Second Document: Document from Chapter 3

| Listing of: \Mylib.Firstdoc(where=(_seqno_=1)) | | |
|--|--------------|------|
| Order by: Insertion | | |
| Number of levels: 1 | | |
| Obs | Path | Type |
| 1 | Univariate#1 | Dir |
| 2 | Freq#1 | Dir |
| 3 | Reg#1 | Dir |

No Current Document: All Documents, showing Labels

| Documents | | |
|-----------|-----------------|-------|
| Obs | Name | Label |
| 1 | Mylib.Backup | |
| 2 | Mylib.Chap7 | |
| 3 | Mylib.Doccopy | |
| 4 | Mylib.Firstcopy | |
| 5 | Mylib.Firstdoc | |

It is possible to execute code on each document in this table. For example, it is possible to submit a LIST statement for each document shown in Output 6.12. This technique is illustrated in the section “Listing Every Document in Your System,” in this chapter.

The DELETE Statement

With the DOC statement out of the way, the chapter returns to folder management. The next statement under discussion is the DELETE statement.

Syntax of the DELETE Statement

```
DELETE path<(where-expression)> <, path-2<(where-expression-2)>>
<, ...path-n<(where-expression-n)>> </ LEVELS= ALL | value>;
```

Before experimenting with the code in this section, use Program 6.12, if you haven't already done so, to make a backup copy of MYLIB.FIRSTDOC into WORK.FIRSTDOC.

As of SAS 9.3, the DELETE statement in the DOCUMENT procedure can delete entire documents as well as folders within documents. To delete an entire document, use the LIBNAME.MEMNAME format *with no backslashes*, as shown in Program 6.27.

Program 6.27: Deleting Entire Documents Before and After SAS 9.3

```
Proc document;
/* 9.3 and later */
DELETE <document name>;
doc;
run;
quit;

/*prior to 9.3 you have to do this */
proc datasets lib=my lib memtype=itemstor;
delete mylib.nb;
quit;
```

Because the DELETE statement can wipe out months of work without prompting you, it is best to make a backup before trying it.

Warning: You Cannot Delete the Current Directory

If you forget which directory is current, you can run into a pitfall when using WHERE= clauses. The WHERE= clause does not change the current directory as the system searches through the tree for objects to be selected. In Program 6.28, for example, the current directory remains the univariate#1 directory (denoted by the caret) throughout the operation. Therefore the delete statement is illegal because the system thinks you are trying to delete the univariate#1 directory.

Program 6.28: Attempt to Delete the Current Directory

```
proc document name=work.firstdoc(write)
  label='Temporary Copy of MYLIB.FIRSTDOC to demonstrate destructive operations';
  copy mylib.firstdoc to ^; ❶
  run;

  dir univariate#1;

  delete ^(where=(_path_ ? '\Age\')) / levels=all; ❷
  run;
  quit;
```

- ❶ Demonstrates the techniques on a copy of important data rather than the original. This line brings in a copy of the mylib.firstdoc folders at the root level.
- ❷ Use the pathname component separator, the \ character, if you want to ensure that you are only working with the keyword of AGE. Otherwise, you might delete paths that contain any word that contains the syllable –age, such as *storage*. That's probably not what you want.

One way to fix Program 6.28 is to set the target directory to something besides the current directory. Program 6.29 illustrates the change that you need to make. The program uses a copy of MYLIB.FIRSTDOC in order to avoid problems. The intent of the original program was to remove all output from the UNIVARIATE procedure that pertained to the variable AGE.

Program 6.29: Corrected Version

```
proc document name=work.firstdoc(write) label='Temporary Copy of MYLIB.FIRSTDOC to demonstrate destructive operations';
  copy mylib.firstdoc to ^;
  run;

  /* current directory is root */
  list ^(where=(_path_ ? '\Age\')) / levels=all;
  run;

  delete univariate(where=(_path_ ? '\Age\')) / levels=all;
  run;

  list / levels=all;
  run;

  quit;
```

Output 6.13 shows that the folders pertaining to the AGE variable were indeed deleted.

Output 6.13: Deleting Using a WHERE= Clause

| | | |
|----|--|-------|
| 1 | \Univariate#1 | Dir |
| 2 | \Univariate#1\ByGroup1#1 | Dir |
| 3 | \Univariate#1\ByGroup1#1\Height#1 | Dir |
| 4 | \Univariate#1\ByGroup1#1\Height#1\Moments#1 | Table |
| 5 | \Univariate#1\ByGroup1#1\Height#1\BasicMeasures#1 | Table |
| 6 | \Univariate#1\ByGroup1#1\Height#1\TestsForLocation#1 | Table |
| 7 | \Univariate#1\ByGroup1#1\Height#1\Quantiles#1 | Table |
| 8 | \Univariate#1\ByGroup1#1\Height#1\ExtremeObs#1 | Table |
| 9 | \Univariate#1\ByGroup2#1 | Dir |
| 10 | \Univariate#1\ByGroup2#1\Height#1 | Dir |
| 11 | \Univariate#1\ByGroup2#1\Height#1\Moments#1 | Table |
| 12 | \Univariate#1\ByGroup2#1\Height#1\BasicMeasures#1 | Table |
| 13 | \Univariate#1\ByGroup2#1\Height#1\TestsForLocation#1 | Table |
| 14 | \Univariate#1\ByGroup2#1\Height#1\Quantiles#1 | Table |
| 15 | \Univariate#1\ByGroup2#1\Height#1\ExtremeObs#1 | Table |

The HIDE Statement

Since the DELETE statement can erase huge amounts of work without any warning, it is worthwhile to discuss alternatives. One alternative to the DELETE statement is the HIDE statement. The HIDE statement excludes a folder from appearing in any output created with the REPLAY statement, which is covered in the previous chapter.

Syntax of the HIDE Statement

```
HIDE path <, path-2, ...path-n>;
```

If the only reason to delete something is that you don't want to see it during replay, consider using the HIDE statement. When you hide a folder, all subfolders are also hidden. Hidden folders are denoted by an asterisk in the type column of the LIST statement output, as shown in Program 6.30.

Program 6.30: Hiding and Its Effect on the LIST Statement

```
ods pdf file='program6_30.pdf';
proc document name=mylib.firstdoc;
  hide reg#2;
  list / levels=2;
run;
quit;
ods pdf close;
```

Output 6.14: User-Hidden Entries

| Listing of: \Mylib.Firstdoc\ | | |
|------------------------------|--------------------------|------|
| Order by: Insertion | | |
| Number of levels: 2 | | |
| Obs | Path | Type |
| 1 | \Univariate#1 | Dir |
| 2 | \Univariate#1\ByGroup1#1 | Dir |
| 3 | \Univariate#1\ByGroup2#1 | Dir |
| 4 | \Freq#1 | Dir |
| 5 | \Freq#1\Table1#1 | Dir |
| 6 | \Reg#1 | Dir |
| 7 | \Reg#1\ByGroup1#1 | Dir |
| 8 | \Reg#1\ByGroup2#1 | Dir |
| 9 | \Reg#2 | Dir* |
| 10 | \Reg#2\MODEL1#1 | Dir* |

* denotes a user-hidden entry.

If you hide a subfolder, there is no indication in the parent level folder that part of that folder's

output will not replay. If you are concerned about hidden output, perform a LIST statement with the / LEVELS=ALL option to uncover any hidden objects. Program 6.31 shows how to hide some output that you might feel you are no longer going to be actively analyzing or presenting. In Program 6.31, this output is the AGE UNIVARIATE output for SEX=F.

Program 6.31: Hiding a Subfolder

```
ods pdf file='program6_31.pdf';
proc document name=my.lib.firstdoc;
hide univariate#1\by group 1#1\age#1;
list / levels=all;
run;
quit;
ods pdf close;
```

Output 6.15 shows the hidden folder by affixing the asterisk to the object type. The Univariate#1 folder could be considered to be partially hidden, since part of it is hidden from replay.

Output 6.15: Hidden Subfolder

| Listing of: \Mylib.Firstdoc\ | | |
|------------------------------|--|--------|
| Order by: Insertion | | |
| Number of levels: All | | |
| Obs | Path | Type |
| 1 | \Univariate#1 | Dir |
| 2 | \Univariate#1\ByGroup1#1 | Dir |
| 3 | \Univariate#1\ByGroup1#1\Height#1 | Dir |
| 4 | \Univariate#1\ByGroup1#1\Height#1\1\Moments#1 | Table |
| 5 | \Univariate#1\ByGroup1#1\Height#1\BasicMeasures#1 | Table |
| 6 | \Univariate#1\ByGroup1#1\Height#1\1\TestsForLocation#1 | Table |
| 7 | \Univariate#1\ByGroup1#1\Height#1\Quantiles#1 | Table |
| 8 | \Univariate#1\ByGroup1#1\Height#1\ExtremeObs#1 | Table |
| 9 | \Univariate#1\ByGroup1#1\Age#1 | Dir* |
| 10 | \Univariate#1\ByGroup1#1\Age#1\1\Moments#1 | Table* |
| 11 | \Univariate#1\ByGroup1#1\Age#1\BasicMeasures#1 | Table* |
| 12 | \Univariate#1\ByGroup1#1\Age#1\1\TestsForLocation#1 | Table* |
| 13 | \Univariate#1\ByGroup1#1\Age#1\Quantiles#1 | Table* |
| 14 | \Univariate#1\ByGroup1#1\Age#1\ExtremeObs#1 | Table* |

This folder is
Hidden!



The UNHIDE statement makes the folder and its contents visible to the replay process. Program 6.32 restores the entire document my.lib.firstdoc to its unhidden state.

Program 6.32: Unhiding a Folder

```
ods pdf file='program6_32.pdf';
proc document name=my.lib.firstdoc;
unhide univariate#1\by group 1#1\age#1;
unhide reg#2;
list reg#2,univariate#1/ levels=3;
run;
quit;
ods pdf close;
```

Output 6.16 shows one of the two restored folders.

Output 6.16: Unhiding a Folder

Listing of: \Mylib.Firstdoc\Reg#2

Order by: Insertion

Number of levels: 3

| Obs | Path | Type |
|-----|---|------|
| 1 | \Reg#2\MODEL1#1 | Dir |
| 2 | \Reg#2\MODEL1#1\Fit#1 | Dir |
| 3 | \Reg#2\MODEL1#1\Fit#1\Weight#1 | Dir |
| 4 | \Reg#2\MODEL1#1\ObswiseStats#1 | Dir |
| 5 | \Reg#2\MODEL1#1\ObswiseStats#1\Weight#1 | Dir |

The hidden status is a permanent part of the folder's identity. If you copy or move a hidden folder, it will remain a hidden folder at its destination, as Program 6.33 demonstrates.

Program 6.33: A Copy of a Hidden Folder Stays Hidden

```
ods html file='program6_33.html';
proc document name=mylib.firstdoc;
hide reg#2;
copy reg#2 to \mylib.chap7\;
run;
quit;

proc document name=mylib.chap7;
list / levels=2;
run;
quit;
ods html close;
```

In Program 6.33 it is not necessary to have two separate calls to the DOCUMENT procedure. With the DOC statement you will learn a method for changing the current document without having to quit and restart the DOCUMENT procedure.

Output 6.17: A Copy of a Hidden Folder Stays Hidden

Listing of: \Mylib.Chap7

Order by: Insertion

Number of levels: 2

| Obs | Path | Type |
|-----|------------------|------|
| 1 | \example#1 | Dir |
| 2 | \example#1\Reg#1 | Dir |
| 3 | \a#1 | Dir* |
| 4 | \a#1\MODEL1#1 | Dir* |

*** denotes a user-hidden entry.**

Notice that the destination of the COPY TO statement required an absolute pathname because the destination was outside the current document.

Extended Examples

The two examples that are presented next showcase the advantage of using the Document Facility's management commands. The first combines the DOC statement and the LIST statement to present a more detailed overview of your entire document collection. The second combines the SETLABEL statement and the COPY TO statements to modify the table of contents that would appear in HTML frame files or PDF files.

Replay with the LEVELS= Option

Since this chapter is mainly concerned with folder management, this section focuses on an application of folder management. In Chapter 5, “The REPLAY Statement,” you were promised a meaningful example of the REPLAY statement with the LEVELS= option. Now that you can create a folder that consists primarily of output objects, and place them at a desired level of your document, it’s time to see a rather creative way of controlling how the table of contents appears.

In her SAS Global Forum 2011 Conference paper, Bari Lawhorn shows how you can change the bookmarks in ODS output by modifying the document’s structure. The programs here show a simple application, and the paper goes beyond the DOCUMENT procedure into the topic of customizing tables of contents. She applies similar methods using the DOCUMENT procedure to customize the table of contents of a PDF file. The paper also presents some interesting facts about tables of contents beyond just what the document can influence. This section is heavily influenced by this paper, and it is used here with permission.

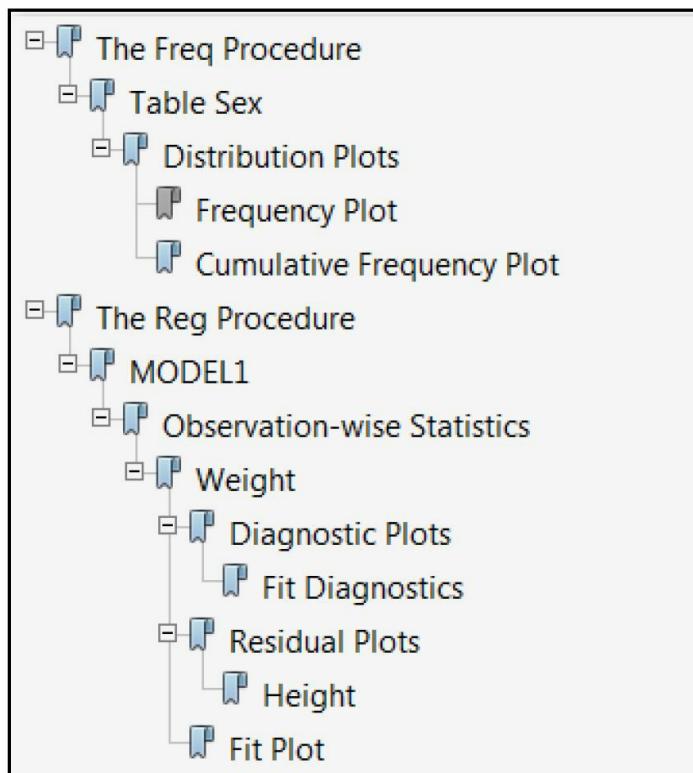
Consider the task of creating a PDF file consisting of all graphical output from your project. Program 6.34 would do the job adequately.

Program 6.34: Displaying All Plots in a Project

```
title 'All Graphics';
ods pdf file="allplots.pdf";
proc document name=mylib.firstdoc;
replay ^(where=(_type_=Graph));
run;
quit;
ods pdf close;
```

Output 6.18 shows the results.

Output 6.18: Table of Contents Derived from Label Path



By reorganizing output into a second document with a flatter organization, the table of contents becomes flatter as well, and perhaps easier to read.

Program 6.35: Setup of a Document Where REPLAY with LEVELS= Is Useful

```

ods pdf file='program6_35.pdf';
proc document name=mylib.firstcopy(write); ①
  copy \mylib.firstdoc(where=(_type_='Graph')) to ^; ②
  run;

list / levels=1;
run;

quit;
ods pdf close;

```

- ① Opens a document in Write mode. Write access mode is helpful when you want to ensure that the only SAS procedure output present in the ODS document is the procedure output produced by this program.
- ② Here is the conditional copy. Note the use of absolute pathnames; this is necessary when the source is in another document.

A common mistake is to omit one of the underscores on document special variables. If you do this, the WHERE= clause will be set to NULL and you will have no output.

After the document is set up, the structure is shown in Output 6.19.

Output 6.19: Document Where Output Objects Are at the Top Level

| Listing of: \Mylib.Firstcopy\ | | |
|-------------------------------|--------------------|-------|
| Order by: Insertion | | |
| Number of levels: 1 | | |
| Obs | Path | Type |
| 1 | FreqPlot#1 | Graph |
| 2 | CumFreqPlot#1 | Graph |
| 3 | DiagnosticsPanel#1 | Graph |
| 4 | ResidualPlot#1 | Graph |
| 5 | FitPlot#1 | Graph |

Program 6.36 includes a few SETLABEL statements to provide more descriptive labels. To keep the code short, only a few labels were changed.

Program 6.36: LEVELS=1 Output

```

ods pdf file="flatterplots.pdf";
proc document name=mylib.firstcopy;
  setlabel freqplot#1 'Bar Plot of Distribution of Variable SEX';
  setlabel cumfreqplot#1 'Cumulative Plot of Variable Sex';
  setlabel diagnosticspanel#1 'Regression Diagnostics for Weight vs. Height, no BY grouping';
  setlabel residualplot#1 'Residuals for Weight vs. Height';
  setlabel fitplot#1 'Fit Plots for Weight vs. Height';
  replay / levels=1;
  run;
  quit;
  ods pdf close;

```

Output 6.20: PDF Table of Contents for LEVELS=1 Output

| |
|---|
| Bar Plot of Distribution of Variable SEX |
| Cumulative Plot of Variable Sex |
| Regression Diagnostics for Weight vs. Height, no BY grouping |
| Residuals for Weight vs. Height |
| Fit Plots for Weight vs. Height |

To the author's eyes, this is much easier to follow, although what is visually helpful to one user might not be to another. You are not limited to one level of folder depth when using this technique. Program 6.37 gives an example of setting up a two-level table of contents with this technique.

Program 6.37: Setting Up a Two-level Hierarchy of Output

```
ods pdf file='program6_37.pdf';
proc document name=mylib.lev2copy(write);
  copy \mylib.firstdoc\freq(where=(_type_='Graph')) to ^;
  copy \mylib.firstdoc\reg(where=(_type_='Graph')) to ^;
  setlabel freq#1 'Frequency Distributions';
  setlabel reg#1 'Linear Model of Height vs. Weight';
  replay / levels=2;
run;

quit;
ods pdf close;
```

Output 6.21 shows the replay. Here there are two folders, each with one level of output.

Output 6.21: Another Example of Managing the Label Path

| |
|-----------------------------------|
| Frequency Distributions |
| Frequency Plot |
| Cumulative Frequency Plot |
| Linear Model of Height vs. Weight |
| Fit Diagnostics |
| Height |
| Fit Plot |

These examples showed how you can customize the level at which your output appears in the table of contents. You can do that by copying it (or moving it) to appear at the desired folder level in your document. Of course, if you don't like the labels, the SETLABEL command can add further customization to your table of contents.

Why does this work? First, the path of an object is part of its metadata, as is the LABEL path, on which the table of contents is based. So, in applications where the label path is made visible, the layout of the output is affected. Here are two examples in which the output's path is visible: the automatic bookmarks of PDF file and the frame file in HTML output. Further, the path is likely to be visible in any destination that creates its table of contents based on the output path.

Listing Every Document in Your System

This section jumps ahead and is a bit more advanced. It assumes some knowledge of the OUTPUT Destination as well as some macro programming. However, material presented in the first six chapters is enough to understand the result of the %alldocs macro. You can download this program from the author's page for this book at support.sas.com/tuchman.

Program 6.38 creates a utility that shows a list of all documents in the system or in a specific library, depending on the LIB= option. This overall list is followed by a listing of each document. The DOC statement prints an enumeration of all documents on your system. For any document, the code 'LIST \docname\' will list the top level folders of the document.

This program is more involved than others you have seen so far. However, because it is useful to do so, it is presented at this point. The technique shown in this program will become more useful as you have to manage more documents. This example is designed to give a quick answer to the question: "What, exactly, do we have stored in this collection of documents?"

The program starts by getting a list of all the documents in the system, and storing this output in a data set. Each element of this data set is the name of a document. Using this data set of document names, the DATA step builds PROC DOCUMENT code that is driven by the information in the data set. To accomplish this, the DATA step uses the CALL EXECUTE routine. CALL EXECUTE builds a block of code that is executed immediately *after* the DATA step terminates. See *SAS Functions and CALL Routines: Reference* for details.

Description of %alldocs Macro Arguments

Levels= The number of levels deep to show a listing of each document. The default is 1.

LIB= A libref that specifies where to look for documents. If the libref is omitted, the macro provides a list of every document in the system is listed.

Program 6.38: %alldocs Macro

```
%macro alldocs(levels=1,lib=);
%* prints a level 1 listing of all documents in the system;

%*****;
%* invoke proc document to get table of all the documents
%* which will be stored in the table DOCLIST
%*****;
proc document;
title "All Documents";
ods output documents=doclist;
ods trace on;
doc %if (&lib ne) %then lib=&lib.; ;
doc close;
run;
ods output close;
quit;
%*****;
%* build a new document procedure call consisting of a
%* LIST statement for each line in the data set DOCLIST
%*****;
data _null_;
length command $512;
set doclist end=nomore;
/* call proc document before beginning work */
if _n_=1 then do;
call execute('proc document;');
end;
/* each line of the data set generates a RUN group of */
/* statements to build a listing with a simple title */
command= cats('list \'name\' / levels=', "&levels.");
call execute ("Title 'Listing of " || name || "'");
call execute(command);
call execute('run;');
/* finish up by closing the document */
if nomore then call execute('run; quit;');
run;
%* once the run is executed, the newly built PROC DOCUMENT
%* code will spring to life;
%mend;

options mprint ;
ods html file='all_listings.html' options(pagebreak='no');
%alldocs()
ods html close;
```

Output 6.22: All Documents and Their Listings, Together

| Documents | | |
|-----------|------------------|------------------------------|
| Obs | Name | Label |
| 1 | Mylib.Backup | |
| 2 | Mylib.Firstcopy | |
| 3 | Mylib.Firstdoc | |
| 4 | Mylib.Graph | |
| 5 | Mylib.Lastone | Analysis for 2004 Model Cars |
| 6 | Mylib.Quickstart | |
| 7 | Mylib.Seconddoc | |
| 8 | Sasuser.Archive | |

| |
|----------------------------|
| Listing of: \Mylib.Backup\ |
| Order by: Insertion |
| Number of levels: 1 |
| Obs Path Type |
| 1 example#1 Dir |
| 2 Univariate#1 Dir |
| 3 Freq#1 Dir |
| 4 Reg#1 Dir |

The output continues with listings of each document in the DOCUMENTS table, with LEVELS set to 1, providing an overview of every document in your session. The remainder of the output is not shown.

Managing ODS Documents via the Graphical User Interface

The Base SAS environment has a GUI for performing some of the file management statements in this chapter. You can use the GUI to create new documents, new folders, and accomplish other tasks.

This section provides an overview to using the GUI. Since the material is already well covered in Andrew Karp's paper as well as Kevin D. Smith's, only a brief comparison of capability is covered here.

Table 6.1: Comparison of GUI and Code Capabilities

| Capability | GUI | Code |
|---|---|------|
| RENAME TO | YES | YES |
| COPY TO | YES | YES |
| MAKE | YES | YES |
| DELETE | YES | YES |
| SETLABEL | <i>Limited</i> to when the object is first created. | YES |
| LINK (see Chapter 10, <i>Working with Links</i>) | NO | YES |
| HIDE | YES | YES |
| NOTE (see Chapter 7, <i>Customizing Output</i>) | NO | YES |
| REPLAY | YES | YES |
| WHERE= clause | NO | YES |

The GUI also includes a code generator. This is an excellent way to learn some of the syntax.

Summary

In this chapter you were shown folder management statements and options needed to customize the order in which your output is stored and, by implication, replayed. The two features making these unique to the DOCUMENT procedure are the WHERE= option, the LEVELS= option, as well as positioning options.

You were also shown two approaches to using document management. One involved copying a document to a flatter format in order to create a neater table of contents. The final example created a report that contained a table of all documents in the system followed by a listing of each.

Chapter 7: Customizing Output

[Introduction](#)

[The Document Used in This Chapter](#)

[Output Objects in an ODS Document](#)

[Introduction to Context](#)

[Example: A Context Reference Chart](#)

[The OBPAGE Statement](#)

[The OBPAGE Statement Might Be Ignored in Some Procedures.](#)

[The OBTITLE Statement](#)

[The SHOW option.](#)

[The OBSTITLE Statement](#)

[Using #BYVAL and #BYVAR Variable Values to Build a Custom Subtitle](#)

[Example: Using the #BYLINE Variable Value](#)

[Object Notes: The OBANOTE and OBBNOTE Statements](#)

[Document Notes](#)

[Adding Notes Using the TEXT Statement](#)

[Adding Notes Using the DOCUMENT Procedure's NOTE Statement](#)

[Document Notes Compared with Object Notes](#)

[The Final Version of the Example Document](#)

[Summary](#)

Introduction

The Document Facility permits you to customize several levels of annotating and footnoting in the document. You can change the titles and footnotes that were originally stored with your output, and you can add other annotations as well. The statements to do this are structured similarly to the global SAS TITLE and FOOTNOTE statements, with which you should be familiar. The document stores with each output object its own list of annotations, called *context*. The chapter defines this term, and shows how titles and footnotes are part of an output object's context.

There are also notes that you can create that are not associated with any output object, but serve to annotate the process as a whole. For example, you can use these types of notes to provide comments marking the transition between segments of your report.

The Document Used in This Chapter

This chapter uses the document CHAP7, which is generated by Program 7.1.

Program 7.1: Data for an Example Report

```
proc document name=mylib.chap7(write);  
  make example;  
  copy \mylib.firstdoc\reg#1(where=  
    (_name_ in ('ParameterEstimates','FitPlot')))  
    to example#1;  
  run;  
  list / levels=all details by groups;  
  run;  
quit;
```

- ❶ The program creates a document in Write access mode so that for this example, you do not need to be concerned with any earlier output changing the sequence numbers.
- ❷ The WHERE= clause was constructed in order to capture the four output objects that are listed in Output 7.1. The DETAILS option and BYGROUPS option of the LIST statement (covered in Chapter 4, “LISTING Documents Using the DOCUMENT Procedure”) create a listing that shows both the page breaks and the names of the BY groups.

Output 7.1: The Four Tables of the Sample Report

| Listing of: \Mylib.Chap7 | | | | | | | | | | |
|--------------------------|---|-------|---------------------|--------------------|--------------------|------------------|-----------------------------|---------------------|---------------|-----|
| Order by: Insertion | | | | | | | | | | |
| Number of levels: All | | | | | | | | | | |
| Obs | Path | Type | Size in Bytes | Created | Modified | Symbolic Link | Template | Label | Page Break | Sex |
| 1 | \example#1\Reg#1 \ParameterEstimates#1 | Table | 291 | 08APR2012:22:53:13 | 08APR2012:22:53:13 | | Stat.REG.ParameterEstimates | Parameter Estimates | | F |
| 2 | \example#1\Reg#1\FitPlot#1 | Graph | 672 | 08APR2012:22:53:13 | 08APR2012:22:53:13 | | Stat.REG.Graphics.Fit | Fit Plot | | F |
| 3 | \example#1\Reg#1 \ParameterEstimates#2 | Table | 291 | 08APR2012:22:53:13 | 08APR2012:22:53:13 | | Stat.REG.ParameterEstimates | Parameter Estimates | | M |
| 4 | \example#1\Reg#1\FitPlot#2 | Graph | 728 | 08APR2012:22:53:13 | 08APR2012:22:53:13 | | Stat.REG.Graphics.Fit | Fit Plot | | M |

Output Objects in an ODS Document

The DOCUMENT procedure statements that begin with OB, such as OBANOTE, OBFOOTN, OBTITLE, and others, work only on output objects. Therefore, it is necessary to review the definition of an output object. An *output object* is one of the following types of objects: GRAPH, TABLE, EQUATION, CROSSTAB, and REPORT. Some items stored in documents that are *not* output objects include LINK, NOTE, and DIR. There might be others as the DOCUMENT procedure continues to evolve. The discussion here focuses on TABLE and GRAPH objects, on the assumption that the concepts presented in this chapter are equally applicable to the others. Program 7.2 shows how to list the output objects in a document.

Program 7.2: Restricting Attention to Output Objects

```
proc document name=mylib.firstdoc;
  list reg#1(where=(_type_ not in ('Dir','Link','Note')) / levels=all;
run;
quit;
```

Recall that the LEVELS=ALL option is necessary. Otherwise, the option would show only the output objects that are stored at the current path. Usually, this is not all of the output, and it might not even produce output. Therefore, use the LEVELS=ALL option when your focus is on output objects. Output 7.2 shows the list of output objects that are stored in the `reg#1` folder.

Output 7.2: Some Output Objects

| Listing of: \Mylib.Firstdoc\Reg#1(where=(_type_ not in ('Dir', 'Link', 'Note'))) | | |
|--|--|-------|
| Order by: Insertion | | |
| Number of levels: All | | |
| Obs | Path | |
| Type | | |
| 1 | \Reg#1\ByGroup1#1\MODEL1#1\Fit#1\Weight#1\NObs#1 | Table |
| 2 | \Reg#1\ByGroup1#1\MODEL1#1\Fit#1\Weight#1\ANOVA#1 | Table |
| 3 | \Reg#1\ByGroup1#1\MODEL1#1\Fit#1\Weight#1\FitStatistics#1 | Table |
| 4 | \Reg#1\ByGroup1#1\MODEL1#1\Fit#1\Weight#1\ParameterEstimates#1 | Table |
| 5 | \Reg#1\ByGroup1#1\MODEL1#1\Obswise Stats#1\Weight#1\DiagnosticPlots#1\DiagnosticsPanel#1 | Graph |
| 6 | \Reg#1\ByGroup1#1\MODEL1#1\Obswise Stats#1\Weight#1\ResidualPlots#1\ResidualPlot#1 | Graph |
| 7 | \Reg#1\ByGroup1#1\MODEL1#1\Obswise Stats#1\Weight#1\FitPlot#1 | Graph |
| 8 | \Reg#1\ByGroup2#1\MODEL1#1\Fit#1\Weight#1\NObs#1 | Table |
| 9 | \Reg#1\ByGroup2#1\MODEL1#1\Fit#1\Weight#1\ANOVA#1 | Table |
| 10 | \Reg#1\ByGroup2#1\MODEL1#1\Fit#1\Weight#1\FitStatistics#1 | Table |
| 11 | \Reg#1\ByGroup2#1\MODEL1#1\Fit#1\Weight#1\ParameterEstimates#1 | Table |
| 12 | \Reg#1\ByGroup2#1\MODEL1#1\Obswise Stats#1\Weight#1\DiagnosticPlots#1\DiagnosticsPanel#1 | Graph |
| 13 | \Reg#1\ByGroup2#1\MODEL1#1\Obswise Stats#1\Weight#1\ResidualPlots#1\ResidualPlot#1 | Graph |
| 14 | \Reg#1\ByGroup2#1\MODEL1#1\Obswise Stats#1\Weight#1\FitPlot#1 | Graph |

Introduction to Context

You already know that the ODS Document Facility saves SAS procedure output for replay without rerunning the original analysis. However, you might also want to save any comments or notable things about that data that you observe. You will likely want to include titles, subtitles, and footnotes, as well as various types of annotations in your output. This output, along with your explanations, is referred to as *context*. Any SAS TITLE_n and FOOTNOTE_n values that were active when the original code was run are saved as part of the output object. The statements described in this chapter enable you to modify these values as well as add additional notes. ODS also provides afternotes and beforenotes, which give you additional opportunity to add textual clarification to your output. All of these context elements are stored as an integral part of the output object by the DOCUMENT procedure.

Another aspect of context that should be included in this conversation is the BY line. You can suppress the normal SAS BY line and customize how BY-group information is displayed using the special commands #BYVAL_n and #BYVAR_n. Examples appear later in the chapter.

Table 7.1 illustrates the order of the object's context.

Table 7.1: Document Context

| |
|--|
| Page break before, using the OBPAGE statement |
| Titles, using the OBTITLE statement |
| Subtitles, using the OBSTITLE statement |
| Before-notes, using the OBBNOTE statement |
| Output table, graph, or report |
| After-notes, using the OBANOTE statement |
| Footnotes, using the OBFOOTN statement |
| Page break after the page after, also using the OBPAGE statement |

Each aspect of context is controlled by a particular DOCUMENT procedure statement. Some context is provided based on the TITLE and FOOTNOTE options that are in place when the SAS procedure output was initially saved in the document.

The statements that modify context are covered from outermost to innermost. Where there is obvious symmetry in the discussion, such as the symmetry between before-notes and after-notes, only one statement is discussed in detail.

Example: A Context Reference Chart

Before jumping into the statements, you might want to have a working copy of each of the context elements. For easy reference, Program 7.3 shows all of the context-related statements that are at your disposal. You can keep this output from this program as your own reference to help you remember the context statements and locations.

Program 7.3: The Object Context

```
ods html file='program7_3.html';
proc document name=example;
  /* copy some output to the current directory */
  copy \mylib.firstdoc\univariate\bygroup1\height to ^;
  dir height;
  /* main work starts here */
  obttitle moments 'Main Title (OBTITLE1 or OBTITLE)';
  obttitle2 moments 'Second Main Title (OBTITLE2)';
  obsubtitle moments 'Subtitle (OBSTITLE1 or OBSTITLE) .A non-null value
    blanks out PROCTITLE name -e.g. "The UNIVARIATE Procedure")';
  obsubtitle2 moments 'A second subtitle (OBSTITLE2)';
  obsubtitle3 moments 'Next comes the BYLINE' / just=l;
  obbnote moments 'First Before Note (OBBNOTE1)';
  obbnote2 moments 'Second Before Note (OBBNOTE2)';
  obanote moments 'First After note (OBANOTE)';
  obanote2 moments 'Second After Note (OBANOTE2)';
  obanote3 moments ':';
  obanote4 moments ':';
  obanote10 moments 'Tenth After Note (OBANOTE10) (can have up to ten of
    each kind of note, just like SAS Titles and Footnotes)';
  obfootn moments 'First Footnote (OBFOOTN)';
  obfootn2 moments 'Second Footnote (OBFOOTN2)';
replay moments;
run;
quit;

ods html close;
```

The information in Output 7.3 is the same as that presented in Table 7.1. The advantage of this format is that you can save this SAS code and print it out when you need a reference to the context elements.

With this basic chart handy, you can now follow along with the discussion of each of these context items in the remainder of the chapter. The emphasis is on taking a few output objects and adding context, and adjusting the page breaks, to make them a cohesive unit.

Output 7.3: Document Context

Main Title (OBTITLE1 or OBITLE)
Second Main Title (OBITLE2)

Subtitle (OBSTITLE) (A non-null value blanks out PROCTITLE name –e.g. ' The UNIVARIATE Procedure')
A second subtitle (OBSTITLE2)

Next comes the BYLINE

Sex=F

First Before Note (OBBNOTE1)
Second Before Note (OBBNOTE2)

| Moments | | | |
|---------------|------------|------------------|------------|
| N | 9 | Sum Weights | 9 |
| Mean | 60.5888889 | Sum Observations | 545.3 |
| Std Deviation | 5.01832752 | Variance | 25.1836111 |

First After note (OBANOTE)
Second After Note (OBANOTE2)

First Footnote (OBFOOTN)
Second Footnote (OBFOOTN2)

To start the extended example, return to the document `mylib.chap7` that you created in Program 7.1. Consider the replay of the `example#1` folder to the PDF destination.

Program 7.4: REPLAY Statement for the Extended Example Folder

```
ods pdf file="program7_4.pdf";
proc document name=mylib.chap7;
replay example;
run;
quit;
ods pdf close;
```

Output 7.4 shows the replayed output. The page breaks need adjustment; ideally, there should be one graph and one table on each page. As the chapter progresses, this output includes more details that make it readable as a report.

Output 7.4: The Starting Replay

program7_4.pdf - Adobe Reader

File Edit View Window Help

1 / 2 63% Tools Sign Comment

Bookmarks

example

- The Reg Procedure
 - Parameter Estimates
 - Fit Plot
- Parameter Estimates
- Fit Plot

The SAS System 22:23 Sunday, April 8, 2012 1

The REG Procedure
Model: MODEL1
Dependent Variable: Weight

Parameter Estimates

| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > t |
|-----------|----|--------------------|----------------|---------|---------|
| Intercept | 1 | -117.36900 | 41.04734 | -2.86 | 0.0244 |
| Height | 1 | 3.42441 | 0.07542 | 5.07 | 0.0014 |

Fit Plot for Weight

Weight

Height

Observations: 9
Parameters: 2
Error DF: 7
MSE: 91.905
R-Square: 0.784
Adj R-Square: 0.754

Pr < 0.05: 95% Confidence Limit: --- 95% Prediction Limit:

The SAS System 22:23 Sunday, April 8, 2012 2

The REG Procedure
Model: MODEL1
Dependent Variable: Weight

Fit Plot for Weight

Weight

Height

Observations: 10
Parameters: 2
Error DF: 8
MSE: 141.177
R-Square: 0.7228
Adj R-Square: 0.685

Pr < 0.05: 95% Confidence Limit: --- 95% Prediction Limit:

The OBPAGE Statement

The outermost element of the context diagram that you can control with the DOCUMENT procedure is the location of page breaks. This can be very important, particularly for PDF files if you don't want to have a separate page for each table, especially if the tables are small. Program 7.5 demonstrates deleting page breaks, but before that must come the definition of OBPAGE statement syntax.

Every output object can have two page breaks associated with it, a *beforepagebreak* and an *afterpagebreak*. You can see the locations of the page breaks without doing a replay. The LIST statement includes this information if you use the DETAILS option, as noted after Program 7.1. You can also delete page breaks with the OBPAGE statement.

Syntax of the OBPAGE Statement

```
OBPAGE output-object / <AFTER> <DELETE>;
```

The AFTER option directs the OBPAGE statement to use the page break after the object, and DELETE deletes the page breaks. The LIST statement with the DETAILS option can show you all of the page breaks in a document. The DELETE option deletes either the before-page break or the after-page break, but not both.

As promised, Program 7.5 demonstrates how to remove the page breaks. The statements from the TABULATE procedure do not necessarily represent the best way to present this data; they are there to demonstrate managing page breaks. In the next chapter, you will see how to take these three tables and combine them into a single table.

Program 7.5: Removing Page Breaks

```
ods document name=mylib.tabulate_test(write);
proc tabulate data=sashelp.class;
var weight height age;
table height*mean;
table age*mean;
table weight * mean;
run;
ods document close;
ods pdf file="program7_5.pdf";
proc document name=mylib.tabulate_test;
dir tabulate(report;
list / details; ①
obpage table#2 / delete; ②
obpage table#3 / delete;
setlabel table#1 'Height Mean';
setlabel table#2 'Age Mean';
setlabel table#3 'Weight Mean';
replay;
run;
quit;
ods pdf close;
```

The far right column in the LIST statement ① output shows that each table has page breaks before them, as seen in Output 7.5a. The OBPAGE statements ② delete the page breaks. Output 7.5b shows that the small tables are all on the same page. As an added measure, ③ the labels were changed to make them more descriptive.

| Listing of: \Mylib.Tabulate_test1 | | | | |
|-----------------------------------|------------------------------|-------|------------------------------|------------|
| Order by: Insertion | | | | |
| Number of levels: All | | | | |
| Obs | Path | Type | Label | Page Break |
| 1 | \Tabulate#1 | Dir | The Tabulate Procedure | |
| 2 | \Tabulate#1\Report#1 | Dir | Cross-tabular summary report | |
| 3 | \Tabulate#1\Report#1\Table#1 | Table | Table 1 | Before |
| 4 | \Tabulate#1\Report#1\Table#2 | Table | Table 2 | Before |
| 5 | \Tabulate#1\Report#1\Table#3 | Table | Table 3 | Before |

Output 7.5b: All Tables Now on Same Page

The SAS System

| Height |
|--------|
| Mean |
| 62.34 |

| Age |
|-------|
| Mean |
| 13.32 |

| Weight |
|--------|
| Mean |
| 100.03 |

There can be at most one page break before and one after an object. Running OPBAGE multiple times does not create additional blank pages.

Program 7.6 shows some more examples of page breaking, this time including afterpage breaks.

Program 7.6: Listing the Page Breaks

```
ods html file='pbtest.html';
title 'Page Breaks';
proc document name=example(write);
  copy \mylib.firstdoc\univariate#\bygroup1\height to ^;
  copy \mylib.firstdoc\univariate#\bygroup2\height to ^;
run;

dir height#1;

obpage moments / after;
```

```

obpage basicmeasures; ②
dir ^^;
dir;
list / levels=all details;
run;
replay;
run;
quit;

ods html close;

```

- ① Adds a page break after the object. OBPAGE_object_; would insert a before-page-break.
- ② The second OBPAGE statement inserts a page break at the start of the BasicMeasures object. The two page breaks combine into one, as you can verify for yourself by replaying the document.

Output 7.6: More Page Breaks

| Listing of: \Work.Example | | | | | | | | | |
|---------------------------|------------------------------|-------|---------------------|--------------------|--------------------|------------------|---------------------------|--|------------------|
| Order by: Insertion | | | | | | | | | |
| Number of levels: All | | | | | | | | | |
| Obs | Path | Type | Size in Bytes | Created | Modified | Symbolic Link | Template | Label | Page Break |
| 1 | \Height#1 | Dir | | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | | Height | |
| 2 | \Height#1\Moments#1 | Table | 619 | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | base.univariate.Moments | Moments | Before and after |
| 3 | \Height#1\BasicMeasures#1 | Table | 526 | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | base.univariate.Measures | Basic Measures of Location and Variability | Before |
| 4 | \Height#1\TestsForLocation#1 | Table | 578 | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | base.univariate.Location | Tests For Location | |
| 5 | \Height#1\Quantiles#1 | Table | 601 | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | base.univariate.Quantiles | Quantiles | |
| 6 | \Height#1\ExtremeObs#1 | Table | 412 | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | base.univariate.ExtObs | Extreme Observations | |
| 7 | \Height#2 | Dir | | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | | Height | |
| 8 | \Height#2\Moments#1 | Table | 621 | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | base.univariate.Moments | Moments | Before |
| 9 | \Height#2\BasicMeasures#1 | Table | 526 | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | base.univariate.Measures | Basic Measures of Location and Variability | |
| 10 | \Height#2\TestsForLocation#1 | Table | 578 | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | base.univariate.Location | Tests For Location | |
| 11 | \Height#2\Quantiles#1 | Table | 601 | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | base.univariate.Quantiles | Quantiles | |
| 12 | \Height#2\ExtremeObs#1 | Table | 412 | 11APR2012:23:25:47 | 11APR2012:23:25:47 | | base.univariate.ExtObs | Extreme Observations | |

The OBPAGE Statement Might Be Ignored in Some Procedures

If a procedure's template has its own NEWPAGE statement, this overrides the page break instruction in the document. In that case, you must disable the NEWPAGE attribute by editing the template. Templates are covered in Chapter 11, "Working with Templates." However, because of the popularity of the SQL procedure, it is mentioned here as well as in the chapter on templates. The pagination issue for the SQL procedure is resolved in Chapter 11.

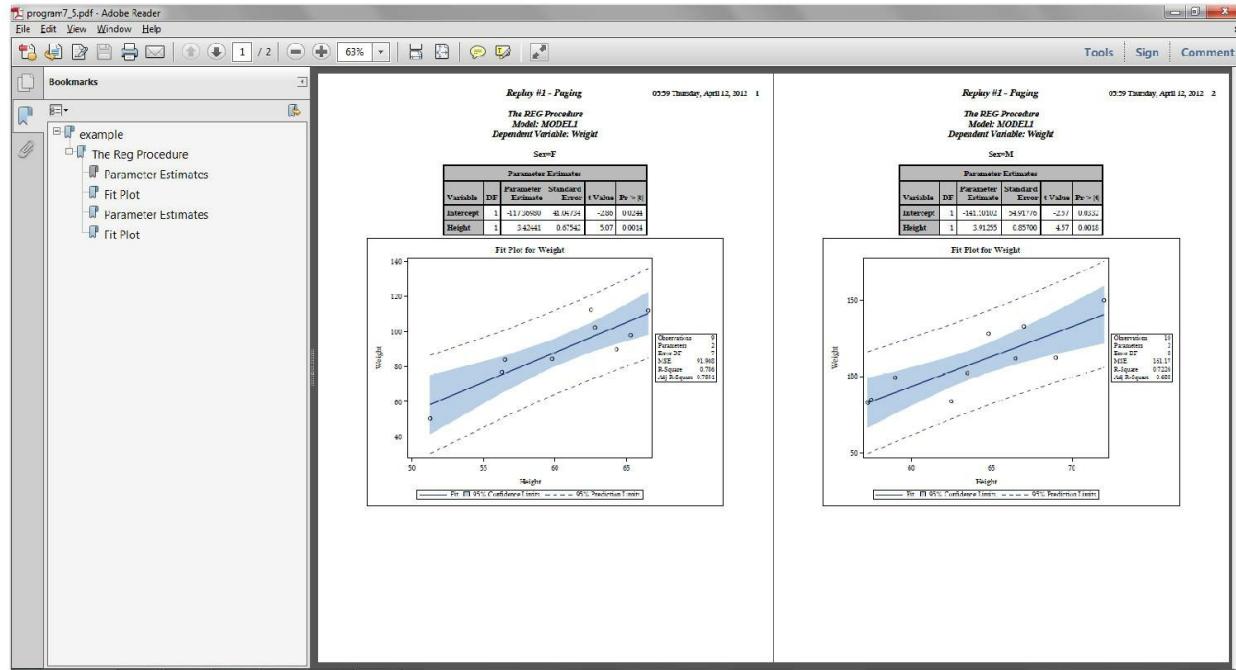
The final example for the OBPAGE statement, Program 7.7, sets up the paging so that there is one graph and one table on each page.

Program 7.7: Adjusting the Pagination

```
ods pdf file="program7_7.pdf";
proc document name=mylib.chap7;
title 'Replay #1 - Paging';
dir example\reg;
obpage parameterestimates;
run;
replay / activetitle;
run;
quit;
ods pdf close;
```

The ACTIVETITLE option ensured that the TITLE statement displays the title that was set for this example instead of the title that was active when you saved the object to the document.

Output 7.7: Modified Page Breaks for the Example Report



The OBITLE Statement

The OBITLE statement modifies the title of the saved output object. The title of the saved object is the value of the TITLE<n> when the object was first sent to the ODS DOCUMENT destination.

Syntax of the OBITLE Statement

```
OBITLE<n> path [ <text>| SHOW]
```

As with the TITLE statement, the default value for *n* is 1, and if you omit the TEXT argument, the saved title is set to empty. Also, as with the global TITLE statement, if you set a null title, you also erase *all titles with a higher number*. Program 7.8 demonstrates how to add some titles to the output object named `Parameterestimates`.

Text Size Limits

<n> can go as high as 10 for the OBITLE, OBSTITLE, OBANOTE, OBBNOTE, and OBFOOTN statements.
The text string may have no more than 32,767 characters.

Examples illustrating the SHOW option begin in a section that is devoted to this topic, after Program 7.11.

Program 7.8: Setting the Titles in the Example Report

```
ods pdf file="program7_8.pdf";
proc document name=my.lib.chap7;
  title 'Replay #2';
  list example\reg / bygroups; ①
  run;

  dir example\reg;
  obpage parameterestimates;
  obtitle1 parameterEstimates#1 "Parameter Estimates"; ②
  obtitle2 parameterEstimates#1 "Basic Model, no interactions";
  obtitle3 parameterEstimates#1 "Normal Error Assumed";
  replay;
  run;
  quit;
  ods pdf close;
```

- ① The first statement of this program is a LIST statement to remind you about how BY-groups are ordered in this folder. Of course, BY lines also communicate this same information.
- ② The OBITLE statements create the updated titles. The titles of the object are shown as Output 7.8. These statements updated the title for only one object – the `parameterestimates` object for SEX=F.

Output 7.8: Updating Titles

Parameter Estimates
Basic Model, no interactions
Normal Error Assumed

The REG Procedure
Model: MODEL1
Dependent Variable: Weight

Sex=F

| Parameter Estimates | | | | | |
|---------------------|----|--------------------|----------------|---------|---------|
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > t |
| Intercept | 1 | -117.36980 | 41.04734 | -2.86 | 0.0244 |
| Height | 1 | 3.42441 | 0.67542 | 5.07 | 0.0014 |

Fit Plot for Weight

If you erase the second title with an OBITLE2 statement, as in Program 7.9, the third title is erased as well, as shown in Output 7.9.

Program 7.9: Deleting Titles

```
ods pdf file="program7_9.pdf";
options nodate;
proc document name=mylib.chap7;
dir example\reg;
obtitle2 parameterestimates#1;
run;
replay;
run;
quit;
ods pdf close;
```

If you want a blank second title, showing the vertical space between the first and third, set it to blank before you define higher numbered titles, as in Program 7.10.

Program 7.10: Blank Titles

```
ods pdf file="program7_10.pdf";
options nodate;
proc document name=mylib.chap7;
title 'Replay #2';
dir example\reg;
obtitle1 parameterEstimates#1 "Parameter Estimates";
setlabel parameterestimates#1 'Parameters: Sex=F';
obtitle2 parameterEstimates#1;
obtitle3 parameterEstimates#1 "Normal Error Assumed";
replay;
run;
quit;
ods pdf close;
```

Output 7.9: Blank Second Title

Parameter Estimates
Normal Error Assumed

If you want to make a one-time change to a title without altering your document, you could use the TITLE<n> statement to set the title, and then use the REPLAY statement with the / ACTIVETITLE

option. This option was mentioned in Chapter 5, “The REPLAY Statement,” without supporting examples. This is the time to demonstrate the option. When you use the / ACTIVETITLE option in the REPLAY statement, you override the stored titles in favor of the current SAS titles. Program 7.11 shows the Active Titles when printing out the plot for SEX=M.

Program 7.11: The ACTIVETITLE Option Uses Global SAS TITLE Values for Replay

```
ods pdf file="program7_11.pdf";
options nodate;
proc document name=my.lib.chap7;
  title1 'Replay #3: Demonstrating the Active Title';
  title2 'Fit Plot for Sex=M only';
  dir example\reg;
  replay fitplot#2 / activetitle;
run;
quit;
ods pdf close;
```

Output 7.10: The Current SAS Titles

Replay #3: Demonstrating the Active Title Fit Plot for Sex=M only

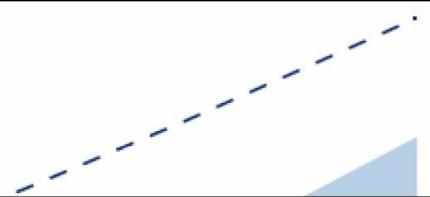
The REG Procedure

Model: MODEL1

Dependent Variable: Weight

Sex=M

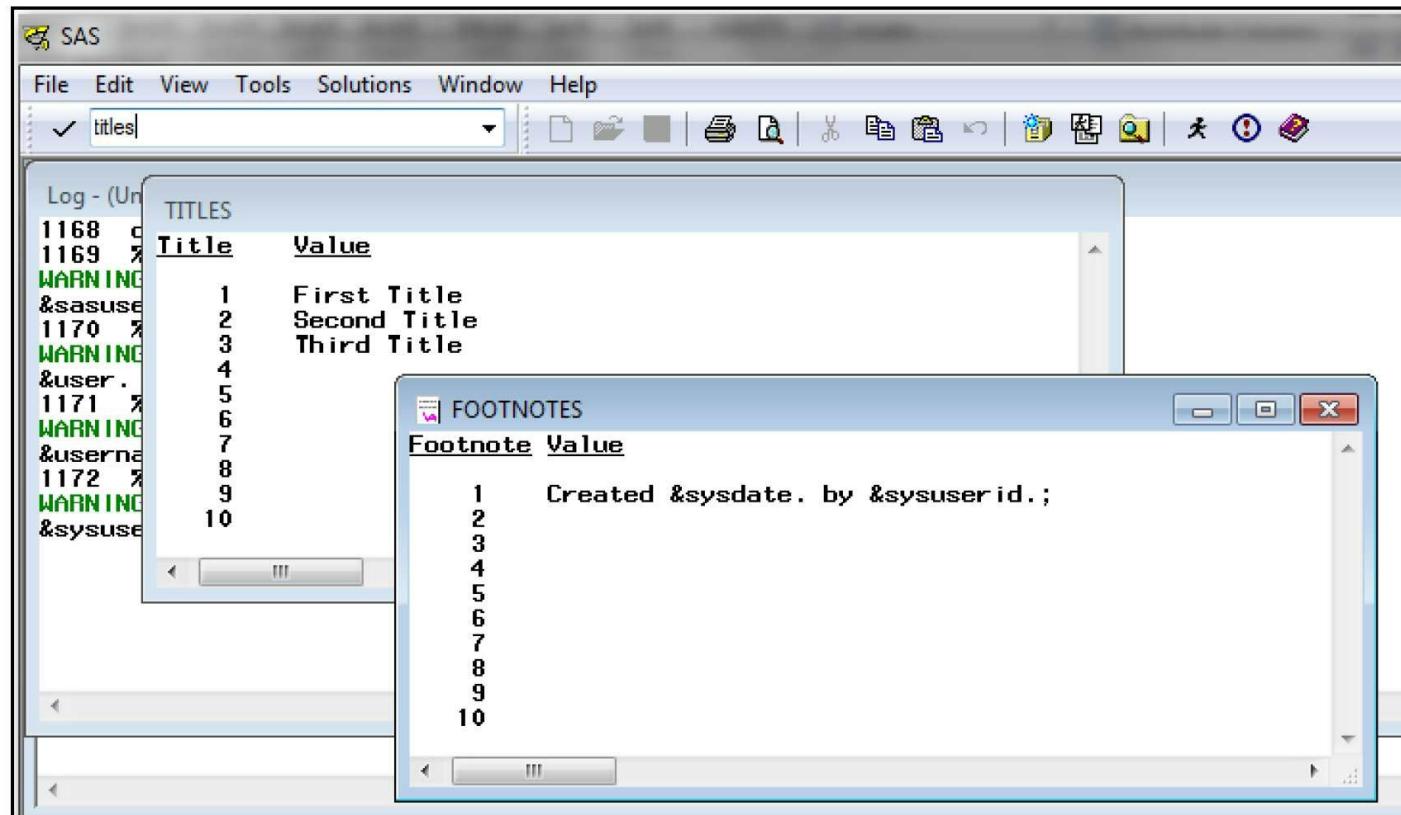
Fit Plot for Weight



The SHOW Option

The SHOW option, new for SAS 9.3, sends a table of the output object contexts to all open destinations. You might already be familiar with the TITLE and FOOTNOTE statements from the SAS windowing environment. These statements display a window containing the values of the current system titles and footnotes. A screen shot of these statements is presented in Display 7.1.

Display 7.1: The TITLE and FOOTNOTE Windows



The SHOW option works in a similar way, printing out the list of titles that are associated with an object. One difference between the SHOW option in the OBTITLE statement and the TITLES window is that the SHOW option simply prints out its table whereas the TITLE statement opens an interactive window. As of SAS 9.3, there is no ability to change the titles interactively. Program 7.12 illustrates the SHOW option with two open destinations.

Even though there is no interactivity with the table of titles that the SHOW option prints, the SHOW option can still be quite useful as a way to edit an object's titles. You can cut and paste your titles into new OBTITLE statements, and rerun them.

To use the SHOW option, use the OBTITLE statement with the SHOW keyword replacing the title text. Program 7.12 demonstrates the SHOW option as it applies to object titles.

Program 7.12: The SHOW Option to OBTITLE

```
footnote "Created &sysdate. by &sysuserid.";
title 'The OBTITLE list as a TABLE using the SHOW option';
ods html file='program7_12.html' style=listing;
proc document name=example;
  dir height;
  obtitle moments show;
run;
ods html close;
```

Output 7.11: Extracting Titles to a Table

The OBTITLEs as a TABLE using the SHOW option

| Context of: \Work.Example\Height#1\Moments#1 | |
|--|----------------------------------|
| Type: Titles | |
| Number | Text |
| 1 | Main Title (OBTITLE1 or OBTITLE) |
| 2 | Second Main Title (OBTITLE2) |

Created 22APR12 by Michael

One application of this table is that you could cut and paste the current OBTITLE values into new OBTITLE statements, and then rerun the OBTITLE statements.

The OBSFOOTN Statement

The OBSFOOTN statement works analogously to the FOOTNOTEn statement in Base SAS. The ACTIVEFOOTN option works for footnotes as ACTIVETITLE works for titles. The rules for blank titles on OBFOOTN footnotes are exactly as they are for OBTITLES. Hence, no additional examples are provided for object footnotes

The OBSTITLE Statement

Object subtitles work similarly to titles. There are a couple of minor differences worth noting. You can adjust the justification of subtitles. Another difference is that there is no SAS global statement affecting subtitles.

Syntax of the OBSTITLE Statement

```
OBSTITLE<n> output-object <'text'> <SHOW></ JUST= LEFT | CENTER | RIGHT>;
```

By default the first SUBTITLE line is the line that says “The *procedure-name* PROCEDURE.” This is called the procedure line, and people often want to omit it from their final reports. Although the ODS NOPROCTITLE option can remove the line with the procedure name, once the output is stored in the document, changing the ODS NOPROCTITLE or PROCTITLE options no longer affects the replay.

Returning to the MYLIB.CHAP7 document, Program 7.13 shows how to remove the line “The REG Procedure” from the printed output. In addition, it performs some other housekeeping with titles that make the final appearance more uniform.

Program 7.13: Customizing Titles and Subtitles

```
ods pdf file='program7_13.pdf';
proc document name=mylib.chap7;
dir example\reg;
obtitle parameterestimates#1 "SASHHELP.CLASS Analysis";
setlabel parameterestimates#1 "Table 1: Parameter Estimates";
obtitle2 parameterestimates#1 "Table 1: Parameter Estimates";

obstitle1 parameterestimates#1 ""; /* could put it back */
obstitle2 parameterestimates#1 "Model: MODEL1";
obstitle3 parameterestimates#1 "Dependent Variable: Weight";
obstitle parameterestimates#1 show;

replay parameterestimates#1;
run;
quit;
ods pdf close;
```

Output 7.12 shows that the procedure line is no longer present, but it could be easily restored by putting it back between the quotation marks on the marked line.

Output 7.12: A Revised First Table from the Example Report

SASHELP.CLASS Analysis
Table 1: Parameter Estimates

Model: MODEL1
Dependent Variable: Weight

Sex=F

| Parameter Estimates | | | | | |
|---------------------|----|--------------------|----------------|---------|---------|
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > t |
| Intercept | 1 | -117.36980 | 41.04734 | -2.86 | 0.0244 |
| Height | 1 | 3.42441 | 0.67542 | 5.07 | 0.0014 |

Using #BYVAL and #BYVAR Variable Values to Build a Custom Subtitle

So far, you can enhance context such as titles and footnotes using standard text strings. If you used BY-group variables in your original analysis, these are also stored with the object. Therefore, BY-variables and BY-values can be used in all context strings.

Every context item, such as footnotes or titles, takes a text argument. If you provide a quoted string, you can use the keywords #BYVAL and #BYVAR to include BY-group information in your object context. Of course, these symbols are valid only if the output object was created as part of BY-group processing. These symbols are analogous to the global symbols of the same name.

If there is more than one BY variable, the symbol *n* stands for the order of the variable when the original BY-group is executed.

As a specific example, consider the analysis in Chapter 1, “Introduction and Quick-Start Guide.” The UNIVARIATE procedure in that analysis used the BY ORIGIN TYPE statement. As a result, Table 7.2 shows which strings the system inserts into your text when you use BYVAR and BYVAL variable values.

Table 7.2: BYVAR and BYVAL

| | |
|---------|---|
| #BYVAR1 | ORIGIN |
| #BYVAL1 | <i>Value of ORIGIN</i> for current object |
| #BYVAR2 | TYPE |
| #BYVAL2 | <i>Value of TYPE</i> |
| #BYLINE | -----ORIGIN=...TYPE=...----- |

Program 7.14 replaces the standard SAS BY line with a custom string using the BY-group information for the stored output. An enhanced HTML destination was included to show that HTML as well as PDF has a contents file.

Program 7.14: Using #BYVAL and #BYVAR Variable Values in Text

```
options nodate nonumber;
ods html file="program7_14.html";
options nobyline; ①
ods html file="program7_14.html"
contents='program7_14_c.html'
frame='program7_14_f.html';
proc document name=mylib.quickstart;
dir univariate;

DIR BYGROUP10#1\MPG_HIGHWAY/* USA SUV*/
obtitle moments "SUV Analysis";
obtitle2 moments "Table 2 : Fuel Economy";
obstitle1 moments "#BYVAL2 from #BYVAL1' / just =l; ②
obstitle2 moments "For this table, BYVAR1=#BYVAR1 and BYVAR2=#BYVAR2"
/ just=L;
replay moments;
list / levels=1 details; ③
run;

run;
quit;
ods html close;
```

① If you plan to use #BYVAL, #BYVAR, or #BYLINE variable values you might find that your

output is less cluttered if you turn off the default BY line.

- ② You can use these '#BY' substitutions in either single or double quoted strings.
- ③ To keep track of which is BYVAR1 and which is BYVAR2, you can either put test output into your code, or you can print a directory with labels. When using BY-groups, SAS sets the label to the BY line as one of several ways to help you remember the correspondence.

Example: Using the #BYLINE Variable Value

There is also a #BYLINE variable value that mimics the appearance of the global BYLINE option. You can use #BYLINE to restore a BY line if the original output was run when OPTIONS NOBYLINE was set.

To print out only the parameters for males, and include the traditional SAS BY line, execute the following inside the PROC DOCUMENT block being used so far.

Program 7.15: Using #BYLINE in a Title

```
ods listing;
options byline;
proc document name=mylib.chap7;
dir example\reg;
obtitle parameterestimates#2 "Plot of Weight vs. height";
obtitle2 parameterestimates#2 "#Byline";
replay parameterestimates#2;
run;
quit;
```

This produces Output 7.13.

Output 7.13: Restoring a BY line

| Plot of Weight vs. height Sex=M | | | | | | 11:37 Sunday, April 22, 2012 |
|------------------------------------|----|--------------------|----------------|---------|---------|------------------------------|
| Parameter Estimates | | | | | | |
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > t | |
| Intercept | 1 | -141.10102 | 54.91776 | -2.57 | 0.0332 | |
| Height | 1 | 3.91255 | 0.85700 | 4.57 | 0.0018 | |

Object Notes: The OBANOTE and OBBNOTE Statements

Until now, all the context statements had direct analogs with SAS global statements, such as TITLE and FOOTNOTE. For example, you saw that the OBTITLE<n> statement is the DOCUMENT procedure's version of global TITLE<n> statements.

Object notes, by contrast, are created and maintained uniquely by the DOCUMENT procedure, and not by any global SAS statement.

Object notes come in two varieties: before-notes, created by the OBBNOTE statement and after-notes covered by the OBANOTE statement. Everything discussed for OBBNOTE applies equally well for the OBANOTE statement. Therefore, in this section, only before-notes are discussed.

After-notes are stored along with the output object. As Table 7.1 showed, they come after the object and before footnotes. The advantage to using object notes over the other types of annotation that are described in this chapter is that they always appear independently of your page break settings. By contrast, object *titles* can sometimes be suppressed by page breaks.

As with other context items, erasing one also erases all others that are subordinate to it (having a higher number). There is also a SHOW option, to print out the current status of the context.

Syntax of the OBANOTE Statement

```
OBANOTE <n> output-object <'text '> <SHOW></ JUST= LEFT | CENTER | RIGHT>;
```

And this applies symmetrically to the OBBNOTE statement as well. Program 7.16 creates another enhancement to the output developed so far in the document mylib.chap7.

Program 7.16: Enhancement to MYLIB.CHAP7 Document Output

```
ods pdf file="program7_16.pdf";
proc document name=mylib.chap7;
dir example\reg;
title «Slope Comparison»;
footnote «Exhibit printed on &SYSDATE.»;
obpage parameterestimates#1 / after delete;
obpage parameterestimates#2 / delete;

obbnote parameterestimates#1 "Females";
obbnote parameterestimates#2 "Males";

obanote parameterestimates#2 "Notice the in this class, males gain 3.9 lbs
per additional inch of ";
obanote2 parameterestimates#2 "height as compared with 3.4 lb/in. for
females";
obanote4 parameterestimates#2 "Comments here , meant to illustrate the
OBANOTE statement, are not " / just=l;
obanote5 parameterestimates#2 "intended to illustrate or recommend proper
statistical procedure" / just=l;
replay parameterestimates#1,parameterestimates#2 / activetitle;
run;
ods pdf close;
```

Output 7.14: Bringing Similar Tables Next to Each Other Facilitates Comparison

Model: MODEL1
Dependent Variable: Weight

Females

| Parameter Estimates | | | | | |
|---------------------|----|--------------------|----------------|---------|---------|
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > t |
| Intercept | 1 | -117.36980 | 41.04734 | -2.86 | 0.0244 |
| Height | 1 | 3.42441 | 0.67542 | 5.07 | 0.0014 |

Males

| Parameter Estimates | | | | | |
|---------------------|----|--------------------|----------------|---------|---------|
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > t |
| Intercept | 1 | -141.10102 | 54.91776 | -2.57 | 0.0332 |
| Height | 1 | 3.91255 | 0.85700 | 4.57 | 0.0018 |

Notice the in this class, males gain 3.9 lbs per additional inch of height as compared with 3.4 lb/in. for females

Comments here , meant to illustrate the OBANOTE statement, are not intended to illustrate or recommend proper statistical procedure

The ACTIVETITLE option was used because the title was created for the purpose of this one printout, and will not be stored with the document. The OBPAGE statements ensure that there is no page break between the tables. Some may find that it is easier to compare tables when they are on the same page as one another. Beforenotes (OBBNOTES) serve well as alternate table headers when the ordinary titles would be suppressed, or when the ordinary titles would be obtrusive. Their use is limited by only your imagination.

The statements that have been discussed so far show how you can change the context of your output without rerunning the original analysis. By managing page breaks and notes, you can save not only output, but your discussions of it, and you can bring comparable outputs closer together on the page. This can be particularly useful for comparing and contrasting the results of similar analyses.

Document Notes

The enhancements and the ability to annotate shown so far in this chapter revolve around output objects. There is another way to customize your output by adding discussion of the output. These are called *notes*, as contrasted with *object-notes*, which have been the discussion so far. Object notes are part of the output object. They do not need to be managed separately. *Notes*, on the other hand, are objects in their own right (but they are not considered *output objects*).

A note is simply a line of text. When you replay the document, the note appears. There are two ways to add document notes to your document. The first method uses the ODS TEXT statement and can be done at the time of the original analysis. The second method uses the DOCUMENT procedure's NOTE statement and can also be done at any time. Document notes can be moved or copied, so you have flexibility about where they are displayed.

You have likely already seen such notes. SAS uses them to update the user on how a calculation is evolving. Examples include Estimates may be subject to bias OR Clustering terminated because no cluster met the conditions for further splitting. However, you are free to add your own.

Adding Notes Using the TEXT Statement

To add notes to the DOCUMENT destination as the computation evolves, specify the text at the time the original code is running. The advantage to this method is that you have access to all your macro variables pertinent to your application. Some of those might be unavailable at replay time.

Syntax of the ODS TEXT Statement

```
ODS TEXT = <text string in single or double quotation marks>\
```

The ODS TEXT statement also has applications outside the ODS document. For more information, see the SAS Help and Documentation.

Program 7.17 illustrates the use of ODS TEXT to report on how many items a query returned.

Program 7.17: Adding Notes to a Document

```
ods document name=mylib.new(write);
title "Reporting on Progress of Computation";
proc sql;
  create table economy as
  select * from sashelp.cars
  where mpg_highway > 35;
quit;
%let rowCnt=&sqlobs.;
/* create a text string conditional on # of observations */
%macro noteit(n);
  %if &n < 30 %then "Result returned fewer than 30 entries (N=&n.). You may
  have too small a sample to make useful inferences";
  %else "Population is (N=&n.)";
%mend;
ods text = %noteit(&rowCnt.);
/* create note whose text depends on the SQL procedure output */
proc univariate data=economy;
var mpg_highway msrp;
run;
ods document close;
proc document name=mylib.new;
  replay;
  run;
quit;
ods html close;
```

Output 7.15 shows the text output, as it occurs before the UNIVARIATE procedure output. You can adjust the style of text for notes if you know how to use the TEMPLATE procedure and something about styles. Since control of styles occurs outside the document, once you learn the appropriate ODS STYLE techniques, you can come back at any time and replay your document with your updated style, thereby changing the appearance of the notes. But there is nothing that you need to do in the document itself.

Output 7.15: Notes Interspersed with SAS Procedure Output

Reporting on Progress of Computation

Result returned fewer than 30 entries (N=24). You may have small sample issues

| Moments | | | |
|-----------------|------------|------------------|------------|
| N | 24 | Sum Weights | 24 |
| Mean | 40.9166667 | Sum Observations | 982 |
| Std Deviation | 6.93395343 | Variance | 48.0797101 |
| Skewness | 2.42323567 | Kurtosis | 6.86456491 |
| Uncorrected SS | 41286 | Corrected SS | 1105.83333 |
| Coeff Variation | 16.9465257 | Std Error Mean | 1.41538732 |

| Basic Statistical Measures | | | |
|----------------------------|----------|---------------------|----------|
| Location | | Variability | |
| Mean | 40.91667 | Std Deviation | 6.93395 |
| Median | 38.00000 | Variance | 48.07971 |
| Mode | 36.00000 | Range | 30.00000 |
| | | Interquartile Range | 6.00000 |

Note: The mode displayed is the smallest of 2 modes with a count of 5.

You can see the note that you created by running a LIST statement, as shown in Program 7.18. Document notes are of type 'Note'.

Program 7.18: Appearance of the Document Note in the Listing

```
ods html file="program7_18.html";
proc document name=mylib.new;
  list / levels=all;
  run;
  quit;
ods html close;
```

Output 7.16: The NOTE as It Appears in the Listings

Reporting on Progress of Computation

| Listing of: \Mylib.New\ | | |
|--------------------------------|---------------|------|
| Order by: Insertion | | |
| Number of levels: All | | |
| Obs | Path | Type |
| 1 | \Text#1 | Note |
| 2 | \Univariate#1 | Dir |

Adding Notes Using the DOCUMENT Procedure's NOTE Statement

It is also possible to add notes within the DOCUMENT procedure. You can do this without rerunning the original analysis.

Syntax of the NOTE Statement

```
NOTE PATH text / positioning argument;
```

After reviewing your output, you might see patterns in the results, or noteworthy comparisons to which you would like to draw attention. By storing these thoughts in the document, they can reside with the results. Thereafter, every time you replay, you will see your comments.

Program 7.19: Inserting and Positioning a Note

```
ods pdf file='program7_19.pdf';
proc document name=mylib.chap7;
dir example;
note mynote "Please consider the environment when printing this document" /
  first just=l;
replay reg;
run;
quit;
ods pdf close;
```

Output 7.17 shows the replay with the note displayed.

Output 7.17: Document Replay with a NOTE in the Top Position of the Document

11:37 Sunday, .

Please consider the environment when printing this document

Females

| Parameter Estimates | | | | | |
|---------------------|----|--------------------|----------------|---------|---------|
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > t |
| Intercept | 1 | -117.36980 | 41.04734 | -2.86 | 0.0244 |
| Height | 1 | 3.42441 | 0.67542 | 5.07 | 0.0014 |

If you no longer want to keep a note, you can delete or hide it. Program 7.20 demonstrates hiding. Deleting works similarly and does not merit a separate example.

Program 7.20: Excluding Notes from Replay with the HIDE Statement

```
ods html file="program7_20.html";
proc document name=mylib.new;
dir univariate\mpg_highway;
hide text#1;
run;
replay;
run;
quit;
ods html close;
```

The replay shown in Output 7.18 no longer displays the note, although it does continue to display the note that SAS generated, “The MODE....” This is actually an after-note, which can be

manipulated using the OBANOTE statement.

Output 7.18: Hiding a Note

| Reporting on Progress of Computation | | | |
|---------------------------------------|------------|------------------|------------|
| Variable: MPG_Highway (MPG (Highway)) | | | |
| Moments | | | |
| N | 24 | Sum Weights | 24 |
| Mean | 40.9166667 | Sum Observations | 982 |
| Std Deviation | 6.93395343 | Variance | 48.0797101 |
| Skewness | 2.42323567 | Kurtosis | 6.86456491 |
| Uncorrected SS | 41286 | Corrected SS | 1105.83333 |
| Coeff Variation | 16.9465257 | Std Error Mean | 1.41538732 |

| Basic Statistical Measures | | | |
|----------------------------|----------|---------------------|----------|
| Location | | Variability | |
| Mean | 40.91667 | Std Deviation | 6.93395 |
| Median | 38.00000 | Variance | 48.07971 |
| Mode | 36.00000 | Range | 30.00000 |
| | | Interquartile Range | 6.00000 |

Note: The mode displayed is the smallest of 2 modes with a count of 5.

The main point of this section is that while ODS TEXT= is fine for sending notes to the document while the program runs, the NOTE statement lets you record your ideas after the program has run, and without having to re-run the original analysis.

One noteworthy aspect of this data is that there seems to be an outlier car that can travel nearly 70 miles to the gallon. Program 7.21 shows you how to make a note of this fact. At the time this book is being written, the price of gas in the US is \$4 per gallon. Interest in high mileage is peaking!

Program 7.21: Inserting Your Own Notes

```
ods html file="program7_21.html";
proc document name=my.lib.new;
  dir univariate\mpg_highway#1;
  note iwantthiscar "Apparently there is a car that
gets 66 miles to the gallon" / before=extremeobs;
  note iwantthis "Need to come back to investigate further. Outlier, or money saver?" / after=iwantthiscar#1;
  replay;
  run;
  quit;
ods html close;
```

This program, in addition to demonstrating notes, is a good opportunity to remember how sequence numbers work for notes. If you are creating a new note, you cannot use a sequence number. If you do, SAS issues an error message iwantthis#1 is an invalid path. In order to create a new note, you need to do so without a sequence number. Once the new note is inserted into the directory, you can use its path as a positioning argument, in the same way as with any other path.

It was not necessary to create both notes with the same name. The author did this to avoid having too many different names for text notes sprinkled throughout the document. However, the pathname of the note is required.

The replay of the relevant part of the document is shown as Output 7.19.

Output 7.19: Extreme Observations with a Comment

Apparently there is a car that gets 66 miles to the gallon

Need to come back to investigate further. Outlier, or money saver?

| Extreme Observations | | | |
|----------------------|-----|---------|-----|
| Lowest | | Highest | |
| Value | Obs | Value | Obs |
| 36 | 23 | 44 | 10 |
| 36 | 14 | 46 | 24 |
| 36 | 6 | 51 | 7 |
| 36 | 5 | 51 | 16 |
| 36 | 4 | 66 | 8 |

Exhibit printed on 14APR12

The note typeface is different from the ODS TEXT= typeface, even though they are both stored as notes in the document.

If you feel comfortable with ODS STYLES and want details, it might help to know that the style for ODS TEXT= notes is UserText. The style for notes created with PROC DOCUMENT is Note.

Document Notes Compared with Object Notes

This chapter introduced two types of notes, document note, and OBANOTES (along with their symmetric cousins, OBBNOTES) but when might you prefer *document notes*?

- When you want to place text between two exhibits, but do not want the text to be associated with a specific table. You have probably already seen notes without realizing it. General comments about the progress of the analysis, such as “Estimation has converged” are implemented as notes when output is saved to the DOCUMENT destination.
- When you sense that you might want to move the discussion to different locations in the replay sequence, such as from after to before an object. This is difficult to do by working with object notes.
- When you want to have a directory populated exclusively with notes that will play as a block of text when replayed (although some touch up formatting might be needed).

Document notes created with the NOTE statement have their own path, and are amenable to file management statements.

When might you prefer *object notes*?

- When the note text makes sense only when in the context of a particular table.
- When you feel sure that you want the text to always accompany the object no matter where it is copied or moved.

The Final Version of the Example Document

Program 7.22 adds the remaining labels and adjusts some titles. Now, the example report is ready to go, either to print out on its own, or to cut and paste into a standard written report.

Program 7.22: Final Report

```
ods pdf file='Final Changes.pdf';
ods html file='final changes.html';
proc document name=my.lib.chap7;
dir example\reg;

setlabel ^ "Height Weight study for SASHELP.CLASS Data"; ①
obtitle parameterestimates#2; ②
obpage parameterestimates#2;
obstitle parameterestimates#2;

obanote4 parameterestimates#2 "Comments here , meant SOLELY to
illustrate the OBANOTE statement,are not";
obanote5 parameterestimates#2 "intended to illustrate or recommend
generally accepted statistical procedure"; ③

/* get the bookmarks right */
setlabel parameterestimates#1 'Table 1: Weight vs. Height for Females';
setlabel fitplot#1 'Graph 1: Line of Best Fit (Females)';
setlabel parameterestimates#2 "Table 2: Weight vs. Height for Males";
setlabel fitplot#2 'Graph 2: Line of Best Fit (Males)'; ④
run;
replay;
run;
quit;
ods pdf close;
ods html close;
```

- ① Setting labels makes the bookmark section easier to read. A label is set both for the main folder example#1 and the main folder reg#1. These labels change the emphasis from the contents of the folder to their purpose.
- ② Inserting a page before the table and graph for SEX=M sets it apart and also makes the titles for parameterestimates#2 visible on the page. Since most of the introductory material was printed on the first page, the second title is limited to reprinting the BY line.
- ③ If you have to change the wording of any context item, you can do so by submitting another OBANOTE, OBBNOTE, OBTITLE, OBSTITLE, or OBFOOTN statement on the existing object to update the text.
- ④ Again, setting the labels makes the document listing and bookmarks, when they appear, more readable. The footnote is not part of the document; it is the active footnote. It was easier to do this as the active footnote so that you do not have to add it to every object.

Output 7.20 shows the result of all the changes.

Output 7.20: Final Version of the Example Document

Bookmarks

example

- Height Weight study for SASHELP.CLASS Data
 - Table 1: Weight vs. Height for Females
 - Graph 1: Line of Best Fit (Females)
 - Table 2: Weight vs. Height for Males
 - Graph 2: Line of Best Fit (Males)

Destination File

| SASHELP.CLASS Analysis | | | | | |
|---|----|--------------------|----------------|---------|---------|
| Table 1: Parameter Estimates | | | | | |
| Model: MODEL1 Dependent Variable: Weight | | | | | |
| Sex=F | | | | | |
| Parameter Estimates | | | | | |
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > t |
| Intercept | 1 | -117.36980 | 41.04734 | -2.86 | 0.0244 |
| Height | 1 | 3.42441 | 0.67542 | 5.07 | 0.0014 |

Notice the in this class, males gain 3.9 lbs per additional inch of height as compared with 3.4 lb/in. for females

Comments here , meant SOLELY to illustrate the OBANOTE statement,are not intended to illustrate or recommend generally accepted statistical procedure

Fit Plot for Weight

Weight

Height

Observations: 9
Parameters: 2
Error DF: 7
MSE: 91.905
R-Square: 0.706
Adj R-Square: 0.7554

Fit 05% Confidence Limits 95% Prediction Limits

Fit Plot for Weight

Weight

Height

Observations: 15
Parameters: 2
Error DF: 13
MSE: 161.17
R-Square: 0.7226
Adj R-Square: 0.689

Fit 05% Confidence Limits 95% Prediction Limits

Please consider the environment before printing

Please consider the environment before printing

Confidential, XYZ Corp, 16APR12

Confidential, XYZ Corp, 16APR12

This is the custom report that has developed throughout this chapter. At this point, not only can you do a basic replay of your SAS procedure output to multiple destinations without rerunning, but you can also build custom reports that would be challenging to build without the document. This is a powerful arsenal to create and manage your output without having to rerun the original analysis. Of course, you can now print this to any destination that supports graphs.

Summary

This chapter showed how you can change the different types of styles, footnotes, and object notes that pertain to an output object. These aspects of output are called *context*. In addition to object notes, the DOCUMENT procedure allows document notes, which are not bound to any particular output object.

Chapter 8: Exporting to Data Sets

Introduction

[The Power of Combining the ODS OUTPUT Destination with the Document Facility](#)

[The ODS OUTPUT Destination](#)

[The ODS OUTPUT Statement](#)

[The ODS OUTPUT Destination without the Document](#)

[The ODS OUTPUT Destination with the Document](#)

[Combining Data Sets from the Document](#)

[Combining the OUTPUT and DOCUMENT Destination](#)

[ODS OUTPUT and the DOCUMENT Procedure's Data Sets](#)

[Simple List Output](#)

[Contents of the Document Data Listing](#)

[Conditionally Executing the DOCUMENT Procedure with CALL EXECUTE](#)

[The SASEDOC Engine](#)

[Accessing a Directory of Output](#)

[Applications of the SASEDOC Engine](#)

[The SASEDOC Engine and Sequence Numbers.](#)

[Summary](#)

Introduction

ODS document output can be converted into SAS data sets. With the capability to export output objects to data sets, you can use traditional DATA step operations to create additional output and build new reports out of old, all the while feeling confident that your new reports will be consistent with those of the past.

For example, if you are handed a document that contains some tables, and you want to create a graph that is based on the data in these tables, use the ODS OUTPUT destination to extract your table to a data set. Then, write SAS ODS graphics code to build the graph. Conversely, a SAS graph can be converted into a table, which can then be used to annotate a graph, or be turned into text that discusses the graph.

After a review of the ODS OUTPUT destination, you will learn how to use the ODS OUTPUT destination to create a data set that brings together analyses from two separate tables.

The SASEDOC LIBNAME engine provides another way to access ODS output that is stored in the document. At the end of the chapter, the two methods are compared and contrasted.

The Power of Combining the ODS OUTPUT Destination with the Document Facility

Sending output from the DOCUMENT procedure to the OUTPUT destination is a powerful tool because you can easily gather all output from a given procedure that is stored in your document into a single data set. This works even if the output is scattered across multiple directories, or even across multiple documents.

Before moving on to the next example, pause for a moment to consider how combining data sets using ODS output and the DOCUMENT procedure might make your life easier. Here are some more scenarios illustrating the two destinations working together:

- You have several graphs that you would like to combine into a single graph. If your graphs were saved using a SAS option that saves the data in a graph, you can recover both series, and combine them into a new graph without closing your SAS session. You would not be able to do so if you just saved your data as a bitmap.
- You have run several analyses and now would like a summary statistic for each analysis. If the output was kept in a path that is named `FitStatistics`, the REPLAY statement would let you export every `fitstatistics` table into a single data set. Contrast this ease of management with the chore of hunting down your data from prior weeks in various spreadsheets, and then cutting and pasting it into a new table.
- For brevity, the examples that were provided demonstrate how to create SAS data set output from a single UNIVARIATE procedure. To envision the possibilities, imagine combining this quarter's results and those of several prior quarters, given only the stored output. What if tracking down the original data sets is impractical or impossible? Fortunately, if you have saved your results in the ODS document, this type of comparison is quite simple.
- You would like a report of sales for the last ten quarters. If each quarter's report was stored in the same document, the REPLAY statement can put them all into a single data set. In fact, this is true even if the reports were stored in different documents.

The ODS OUTPUT Destination

This section is a discussion of the ODS OUTPUT destination. This might be a review for some readers, but the discussion is presented because the book does not assume that the reader has ODS OUTPUT experience.

The first examples provide illustrations of the concepts using a sample program. The illustrations connect the program to the different aspects of ODS.

The ODS OUTPUT Statement

Here is the syntax of the ODS OUTPUT statement:

```
ODS OUTPUT data-set-definition(s);
```

Here is the syntax for *data-set-definition*:

```
output-object-specification(<MATCH_ALL>) = data set;
```

An output object specification, for the time being, will simply be the ODS name. Recall that all ODS output has a name, as discussed in Chapter 2, “The ODS DOCUMENT Destination.” There are several articles and books that give a full discussion of every option of the ODS OUTPUT destination. Consult *Output Delivery System: The Basics and Beyond* or the *Output Delivery System: User’s Guide: Second Edition*.

The ODS OUTPUT Destination without the Document

With the ODS OUTPUT statement defined, it is time to illustrate the process with a program and a diagram.

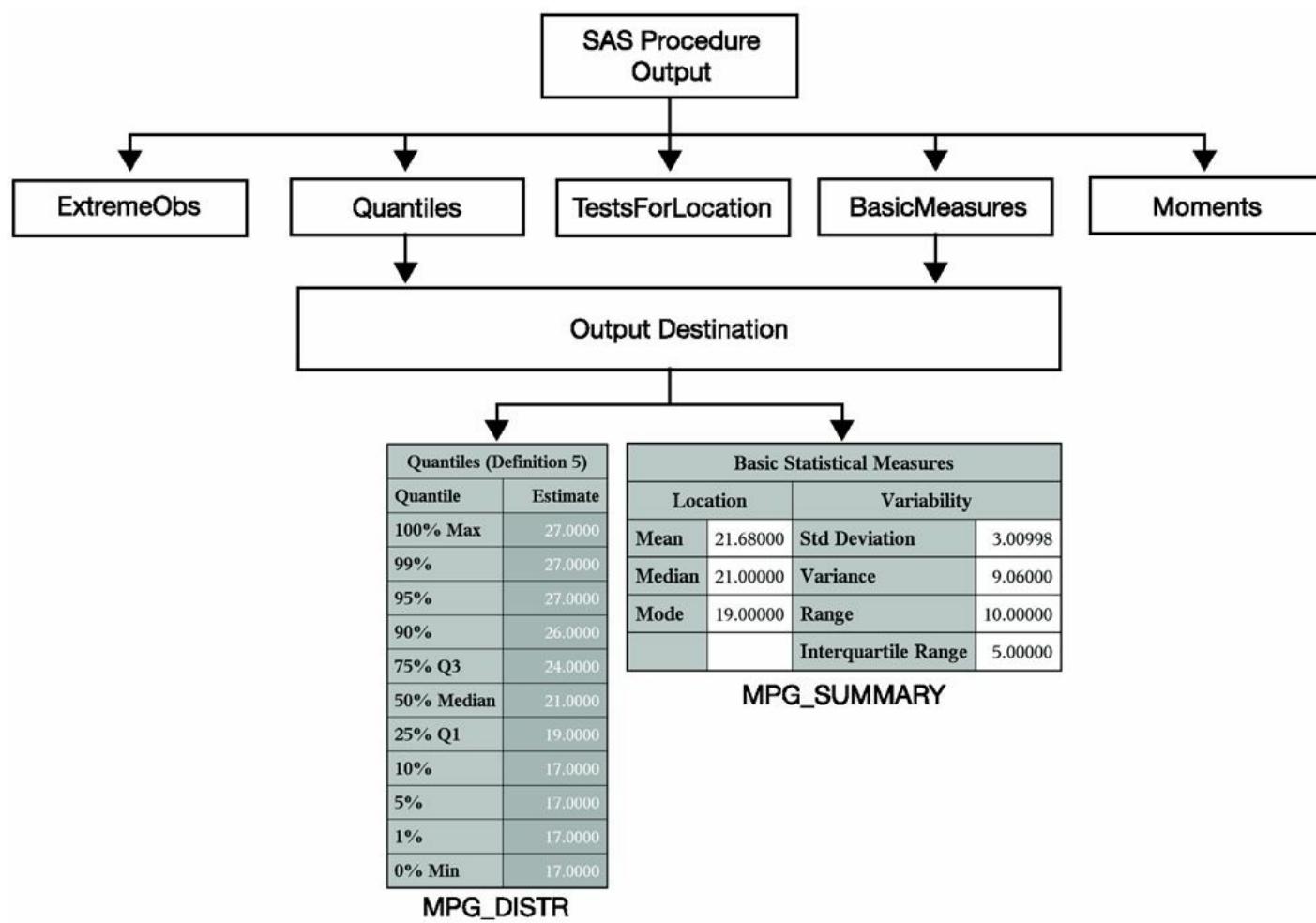
The ODS OUTPUT destination sends SAS output to a data set. The OUTPUT destination requires a separate data set name for each type of object that you want saved. This means that you need to know the ODS name of each output you want to send to the OUTPUT destination. You also need to decide the name of the data set that will store that data.

Program 8.1: Saving Objects to Data Sets Using the OUTPUT Destination

```
ods output quantiles=distrs  
      basicMeasures=summaries; ①  
proc univariate data=sashelp.cars;  
  where type='SUV';  
  var mpg_highway;  
run;  
ods output close; ②
```

The ODS OUTPUT statement ① tells the destination to take every `quantiles` object it sees while the destination is open, and inserts that data into the data set `distrs`. Similarly, ODS saves all `BasicMeasures` tabular output to a single data set named `Summaries`. When the destination is closed ②, the data sets are created. The process is shown in Figure 8.1.

Figure 8.1: The OUTPUT Destination without the Document



If you submit more than one UNIVARIATE procedure while the ODS OUTPUT destination is open, the data set created would include that `basicmeasures` and `quantiles` output from that procedure's

output as well.

Program 8.2: Combining Output

```
/*from program 1.1 */  
proc sort data=sashelp.cars out=WORK.cars;  
  where upcase(type) ne 'HYBRID';  
  by origin type;  
run;  
ods output basicmeasures=b quantiles=q;  
proc univariate data=cars;  
  by origin type;  
  var mpg_highway mpg_city;  
run;
```

Output 8.1 shows how the `basicmeasures` and `quantiles` objects have been combined into a single data set. Since the data set consists of several objects, you need a way to determine the origin of each section of the data set. If you used BY groups in the original analysis, the BY-groups variables are on the data set. Other variables that determine the output's contents, such as the variable used for the VAR statement, are also included.

If you run Program 8.2 immediately after Program 8.1, your original output would not be lost. Program 8.2 saves the output objects to different names than Program 8.1 does.

Table 8.1: Data Sets Created by Programs 8.1 and 8.2

| Data Set Name | Contents |
|----------------|--|
| WORK.DISTRYS | Quantiles output (SUV, MPG_HIGHWAY) |
| WORK.SUMMARIES | Basicmeasures Output |
| WORK.B | Basicmeasures output for all by Groups |
| WORK.Q | Quantiles output for all BY-groups |

So long as you choose names that do not conflict with previous names, your output will not be overwritten.

Output 8.1: Combined BasicMeasures Output

| Obs | Origin | Type | VarName | LocMeasure | LocValue | VarMeasure | VarValue |
|-----|--------|-------|-------------|------------|----------|---------------------|----------|
| 1 | Asia | SUV | MPG_City | Mean | 17.32000 | Std Deviation | 2.76466 |
| 2 | Asia | SUV | MPG_City | Median | 17.00000 | Variance | 7.64333 |
| 3 | Asia | SUV | MPG_City | Mode | 17.00000 | Range | 9.00000 |
| 4 | Asia | SUV | MPG_City | | _ | Interquartile Range | 4.00000 |
| 5 | Asia | SUV | MPG_Highway | Mean | 21.68000 | Std Deviation | 3.00998 |
| 6 | Asia | SUV | MPG_Highway | Median | 21.00000 | Variance | 9.06000 |
| 7 | Asia | SUV | MPG_Highway | Mode | 19.00000 | Range | 10.00000 |
| 8 | Asia | SUV | MPG_Highway | | _ | Interquartile Range | 5.00000 |
| 9 | Asia | Sedan | MPG_City | Mean | 22.84043 | Std Deviation | 4.93899 |
| 10 | Asia | Sedan | MPG_City | Median | 21.00000 | Variance | 24.39362 |
| 11 | Asia | Sedan | MPG_City | Mode | 18.00000 | Range | 20.00000 |
| 12 | Asia | Sedan | MPG_City | | _ | Interquartile Range | 7.00000 |

Output 8.2: The Quantiles Output

Combining Data Sets using the OUTPUT Destination

| Obs | Origin | Type | VarName | Quantile | Estimate |
|-----|--------|------|-------------|------------|----------|
| 1 | Asia | SUV | MPG_Highway | 100% Max | 27.0 |
| 2 | Asia | SUV | MPG_Highway | 99% | 27.0 |
| 3 | Asia | SUV | MPG_Highway | 95% | 27.0 |
| 4 | Asia | SUV | MPG_Highway | 90% | 26.0 |
| 5 | Asia | SUV | MPG_Highway | 75% Q3 | 24.0 |
| 6 | Asia | SUV | MPG_Highway | 50% Median | 21.0 |
| 7 | Asia | SUV | MPG_Highway | 25% Q1 | 19.0 |
| 8 | Asia | SUV | MPG_Highway | 10% | 17.0 |
| 9 | Asia | SUV | MPG_Highway | 5% | 17.0 |
| 10 | Asia | SUV | MPG_Highway | 1% | 17.0 |
| 11 | Asia | SUV | MPG_Highway | 0% Min | 17.0 |
| 12 | Asia | SUV | MPG_City | 100% Max | 22.0 |
| 13 | Asia | SUV | MPG_City | 99% | 22.0 |
| 14 | Asia | SUV | MPG_City | 95% | 22.0 |
| 15 | Asia | SUV | MPG_City | 90% | 21.0 |

While outside the scope of this chapter, the ODS OUTPUT destination has additional options to explore, such as controlling whether output is saved to a single large data set or to many smaller ones.

The ODS OUTPUT Destination with the Document

When you use the REPLAY statement to send procedure output that is stored in the document to the OUTPUT destination, the ODS OUTPUT destination can create the same data sets that are created when the procedure output is sent directly to the OUTPUT destination.

Program 8.3 shows how the process works when the document is involved. As with the discussion in Chapter 2, “The ODS DOCUMENT Destination,” there is an intermediate step involved in saving the SAS procedure output to the document.

Program 8.3: ODS OUTPUT and ODS DOCUMENT Together

```
ods output quantiles=q  
  basicMeasures=b; ❶  
proc document name=mylib.quickstart;  
  replay / dest=output; ❷  
  run;  
  quit;  
ods output close; ❸
```

Program 8.3 produces no printable output. It does, however, create the two data sets that are excerpted as Output 8.3.

Output 8.3a: Replay of BasicMeasures WORK.B Output to Data Set

| Type | VarName | Measure of Location | Value of Measure of Location | Measure of Variability | Value of Measure of Variability |
|------|---------|---------------------|------------------------------|------------------------|---------------------------------|
| SUV | MSRP | Mean | 29569.0 | Std Deviation | 11843 |
| SUV | MSRP | Median | 27560.0 | Variance | ***** |
| SUV | MSRP | Mode | . | Range | 47637 |
| SUV | MSRP | | - | Interquartile Range | 13391 |
| SUV | Invoice | Mean | 26916.5 | Std Deviation | 9965 |
| SUV | Invoice | Median | 24843.0 | Variance | 99294671 |
| SUV | Invoice | Mode | . | Range | 39506 |
| SUV | Invoice | | - | Interquartile Range | 11554 |

Output 8.3b: Quantiles WORK.Q Output to Data Set

| | Origin | Type | VarName | Quantile | Estimate |
|----|--------|------|---------|------------|----------|
| 1 | Asia | SUV | MSRP | 100% Max | 64800 |
| 2 | Asia | SUV | MSRP | 99% | 64800 |
| 3 | Asia | SUV | MSRP | 95% | 54765 |
| 4 | Asia | SUV | MSRP | 90% | 45700 |
| 5 | Asia | SUV | MSRP | 75% Q3 | 33840 |
| 6 | Asia | SUV | MSRP | 50% Median | 27560 |
| 7 | Asia | SUV | MSRP | 25% Q1 | 20449 |
| 8 | Asia | SUV | MSRP | 10% | 18892 |
| 9 | Asia | SUV | MSRP | 5% | 18690 |
| 10 | Asia | SUV | MSRP | 1% | 17163 |
| 11 | Asia | SUV | MSRP | 0% Min | 17163 |
| 12 | Asia | SUV | Invoice | 100% Max | 56455 |
| 13 | Asia | SUV | Invoice | 99% | 56455 |

Figure 8.2 shows the process behind this program. As with Program 8.1, the ODS OUTPUT statement ❶ in Program 8.3 declares which ODS output will be turned into data sets, based on the

ODS name as well as on the name of the SAS data set that will hold that output.

The REPLAY statement now uses the / DEST=OUTPUT ② option to direct output to the data OUTPUT destination. The name of the file that is created is in the ODS output statement. The output data set is *not* named `output`. The OUTPUT destination is waiting offstage for the specific output objects for which it is listening. Strictly speaking, it is not necessary to include this option because the DOCUMENT procedure will send output to all open destinations. In this instance, only the OUTPUT destination is of interest. Thus, when you use the DEST=OUTPUT option, only the OUTPUT destination receives the output.

It is always good practice to close any destination when you are finished writing to it. The OUTPUT destination is no exception. ③

Figure 8.2 differs from Figure 8.1 in that the UNIVARIATE procedure output has been replaced with a REPLAY statement from the DOCUMENT procedure.

Figure 8.2: ODS with the DOCUMENT

Input Document

Listing of: \Mylib.Quickstart\outreplay#1

Order by: Insertion

Number of levels: All

| Path | Type |
|--|-------|
| \outreplay#1\Univariate#1 | Dir |
| \outreplay#1\Univariate#1\MPG_Highway#1 | Dir |
| \outreplay#1\Univariate#1\MPG_Highway#1\Moments#1 | Table |
| \outreplay#1\Univariate#1\MPG_Highway#1\BasicMeasures#1 | Table |
| \outreplay#1\Univariate#1\MPG_Highway#1\TestsForLocation#1 | Table |
| \outreplay#1\Univariate#1\MPG_Highway#1\Quantiles#1 | Table |
| \outreplay#1\Univariate#1\MPG_Highway#1\ExtremeObs#1 | Table |



REPLAY/DEST=OUTPUT



Output Destination Waits for BasicMeasures and Quantiles output objects to be produced.



| Quantile | Estimate |
|------------|----------|
| 100% Max | 27.0000 |
| 99% | 27.0000 |
| 95% | 27.0000 |
| 90% | 26.0000 |
| 75% Q3 | 24.0000 |
| 50% Median | 21.0000 |
| 25% Q1 | 19.0000 |
| 10% | 17.0000 |
| 5% | 17.0000 |
| 1% | 17.0000 |
| 0% Min | 17.0000 |

| Basic Statistical Measures | | | |
|----------------------------|----------|---------------------|----------|
| Location | | Variability | |
| Mean | 21.68000 | Std Deviation | 3.00998 |
| Median | 21.00000 | Variance | 9.06000 |
| Mode | 19.00000 | Range | 10.00000 |
| | | Interquartile Range | 5.00000 |

MPG_SUMMARY

MPG_DISTR

Combining Data Sets from the Document

Now that you have seen how to create data sets from procedure output that is stored in the document, you should also realize that you can do anything with those data sets that you can do with DATA step code.

Here is a situation that would be difficult to accomplish without the ODS document. Assume that the original data set for this analysis, WORK.CARS, was missing. Furthermore, what if you had no practical way to restore the original data set or run the original analysis code? Now, what if you are asked to compare figures that would ordinarily be kept in separate tables? For example, how does the median of Miles per Gallon compare for cars in different countries and types? With the ODS document, this is easy. You can replay document objects to the OUTPUT destination data set and extract the records that pertain to the median. Program 8.4 shows how this is done:

Program 8.4: Combining Reports Using the OUTPUT Destination

```
ods output basicmeasures=b quantiles=q moments=m;
title "Combining Data Sets using the OUTPUT Destination";
proc document name=my.lib.quickstart;
  replay univariate (where=(_path_ ? 'MPG')) / dest=output ;
  run;
quit;
ods output close;
```

Output 8.4 shows excerpts from each table. Once the OUTPUT destination creates your data in tabular form, you can combine them into new exhibits. The remaining programs in this section illustrate one application, and you can imagine many others. Although the `moments` object is not used again, it is kept because it frequently has handy reference information, such as the number of cars in each class.

Output 8.4: Output Tables Used to Build Combined Exhibit

Moments

| Obs | Origin | Type | VarName | Label1 | cValue1 | nValue1 | Label2 | cValue2 | nValue2 |
|-----|--------|------|-------------|-----------------|------------|-------------|------------------|------------|------------|
| 1 | Asia | SUV | MPG_City | N | 25 | 25.000000 | Sum Weights | 25 | 25.000000 |
| 2 | Asia | SUV | MPG_City | Mean | 17.32 | 17.320000 | Sum Observations | 433 | 433.000000 |
| 3 | Asia | SUV | MPG_City | Std Deviation | 2.76465791 | 2.764658 | Variance | 7.64333333 | 7.643333 |
| 4 | Asia | SUV | MPG_City | Skewness | 0.12662092 | 0.126621 | Kurtosis | -0.8227381 | -0.822738 |
| 5 | Asia | SUV | MPG_City | Uncorrected SS | 7683 | 7683.000000 | Corrected SS | 183.44 | 183.440000 |
| 6 | Asia | SUV | MPG_City | Coeff Variation | 15.9622281 | 15.962228 | Std Error Mean | 0.55293158 | 0.552932 |
| 7 | Asia | SUV | MPG_Highway | N | 25 | 25.000000 | Sum Weights | 25 | 25.000000 |

Basic Measures

| Obs | Origin | Type | VarName | LocMeasure | LocValue | VarMeasure | VarValue |
|-----|--------|------|-------------|------------|----------|---------------------|----------|
| 1 | Asia | SUV | MPG_City | Mean | 17.32000 | Std Deviation | 2.76466 |
| 2 | Asia | SUV | MPG_City | Median | 17.00000 | Variance | 7.64333 |
| 3 | Asia | SUV | MPG_City | Mode | 17.00000 | Range | 9.00000 |
| 4 | Asia | SUV | MPG_City | | - | Interquartile Range | 4.00000 |
| 5 | Asia | SUV | MPG_Highway | Mean | 21.68000 | Std Deviation | 3.00998 |

Quantiles

| Obs | Origin | Type | VarName | Quantile | Estimate |
|-----|--------|------|----------|------------|----------|
| 1 | Asia | SUV | MPG_City | 100% Max | 22.0 |
| 2 | Asia | SUV | MPG_City | 99% | 22.0 |
| 3 | Asia | SUV | MPG_City | 95% | 22.0 |
| 4 | Asia | SUV | MPG_City | 90% | 21.0 |
| 5 | Asia | SUV | MPG_City | 75% Q3 | 19.0 |
| 6 | Asia | SUV | MPG_City | 50% Median | 17.0 |
| 7 | Asia | SUV | MPG_City | 25% Q1 | 15.0 |
| 8 | Asia | SUV | MPG_City | 10% | 13.0 |
| 9 | Asia | SUV | MPG_City | 5% | 13.0 |
| 10 | Asia | SUV | MPG_City | 1% | 13.0 |
| 11 | Asia | SUV | MPG_City | 0% Min | 13.0 |

Program 8.5 reshapes and sorts the data so that the variables can be compared by country. The program renames `type` to `vehicle_type` in order to make the variable names more specific.

Program 8.5: Reshaping the Data

```

data medians;
length mileage_type $8; /* to avoid truncation */
set q;
where quantile='50% Median';
mileage_Type=substr(varname,5); /* remove MPG_ */
drop varname;
rename type=vehicle_type; /* to clarify variable names */
run;

proc sort data=medians;
by mileage_type vehicle_type origin;
run;

```

The `medians` data set is shown in Output 8.5.

Output 8.5: The Reshaped Data

| Obs | mileage_type | vehicle_type | Origin | Estimate |
|-----|--------------|--------------|--------|----------|
| 1 | City | SUV | Asia | 17.0 |
| 2 | City | SUV | Europe | 14.5 |
| 3 | City | SUV | USA | 15.0 |
| 4 | City | Sedan | Asia | 21.0 |
| 5 | City | Sedan | Europe | 19.0 |
| 6 | City | Sedan | USA | 20.0 |
| 7 | City | Sports | Asia | 20.0 |
| 8 | City | Sports | Europe | 18.0 |
| 9 | City | Sports | USA | 17.0 |
| 10 | City | Truck | Asia | 16.5 |
| 11 | City | Truck | USA | 15.5 |
| 12 | City | Wagon | Asia | 21.0 |
| 13 | City | Wagon | Europe | 19.0 |
| 14 | City | Wagon | USA | 22.0 |
| 15 | Highway | SUV | Asia | 21.0 |
| 16 | Highway | SUV | Europe | 19.0 |

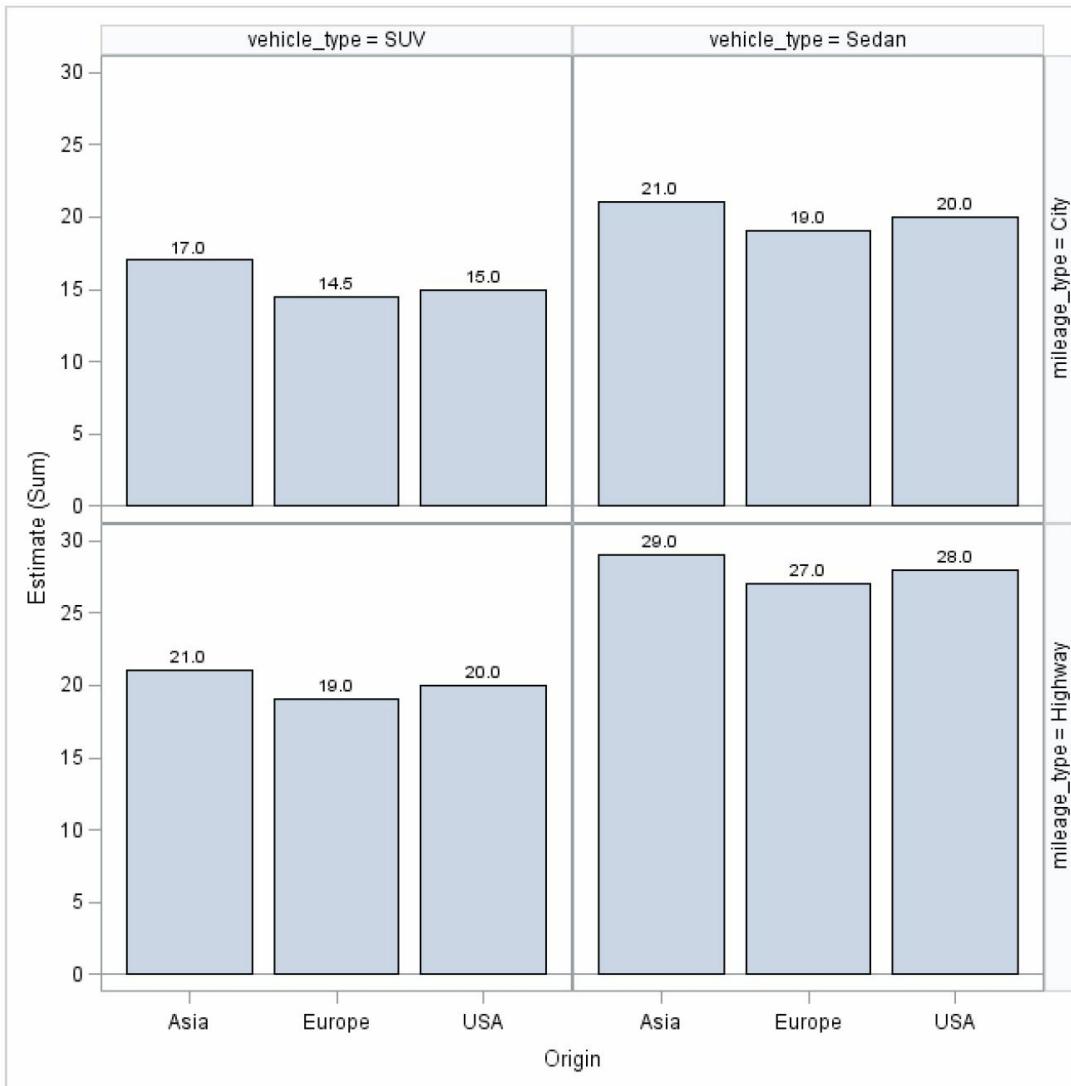
Program 8.6 continues the extended example by building a combined graph based on the `medians` data.

Program 8.6: Building a Graph from Reshaped Data

```
ods html file="program8_6.html";
options device=png;
proc sgpanel data=medians;
  panelby vehicle_type mileage_type / layout=lattice columns=2;
  vbar origin / response=estimate datalabel=estimate;
run;
ods html close;
```

If you want to make this graph even more sophisticated (for example, by including other features of the SGPROC procedure), you could always do so. The Document Facility makes it easier to experiment with output appearance because your output is always waiting for you, available with a REPLAY statement.

Output 8.6: An Excerpt of the Graphs



To make additional annotations, you could put this graph back into your document, and use all of the annotation techniques to give the document additional context, such as updated footnotes, beforenotes, afternotes, or document notes.

Program 8.7 shows how to create these new exhibits, stored in their own folder in `mylib.quickstart`. Although this program is essentially Program 8.6 with different header information, it is included as a review of existing material. Labels were added to make the output from the LIST statement easier to read, and both before- and after-notes create additional insight into the graphical output. Furthermore, this output remains available to you in folder `new_exhibits` for replay at any time. Now, in addition to just “output,” you have a presentation combining data with insight. You can easily add a replay of this presentation, conveniently stored in a separate folder, to the output that you present to your audience.

Program 8.7: Output Stored and Annotated

```

ods html file="program8_7.html";
ods graphics / width=3.5in height=3in;
proc document name=mylib.quickstart;
  dir new_exhibits;
  dir sgpanel;
  setlabel sgpanel#1 'SUV and Sedan';
  setlabel sgpanel#2 'Sports and Truck';
  setlabel sgpanel#3 'Wagons';
  list / details;
  obbbnote sgpanel#1 'Analyses of City MPG by Origin and Type';
  obanote sgpanel#2 'Notice that USA has lowest MPG in City Mileage for
    Sports Cars';

```

```
obanote spanel#3 'Notice that USA has highest MPG in City Mileage for  
Wagons';
```

```
run;  
replay;  
run;  
quit;  
ods html close;
```

Output 8.7 shows the enhanced listing and the excerpts from the program's output. The graphs were presented in Output 8.6.

Output 8.7: Listing and Revised Output Showing Notes Added to Graphs

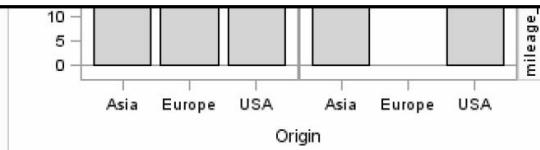
Listing of Graphs in this example

Listing of: \Mylib.Quickstart\new_exhibits#1\Spanel#1

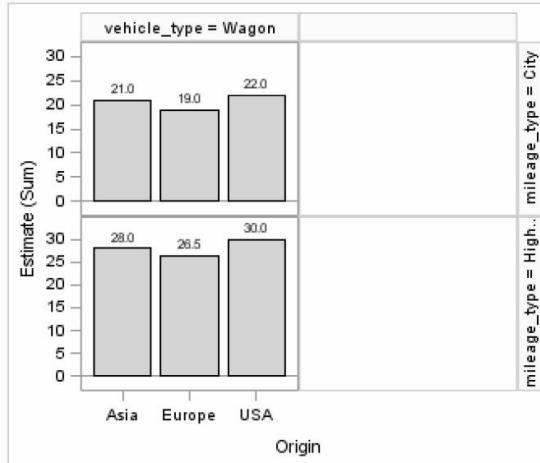
Order by: Insertion

Number of levels: 1

| Obs | Path | Type | Label | Page Break |
|-----|------------------------------------|-------|------------------|------------|
| 1 | \new_exhibits#1\Spanel#1\SGPanel#1 | Graph | SUV and Sedan | Before |
| 2 | \new_exhibits#1\Spanel#1\SGPanel#2 | Graph | Sports and Truck | |
| 3 | \new_exhibits#1\Spanel#1\SGPanel#3 | Graph | Wagons | |



Notice that USA has lowest MPG in City Mileage for Sports Cars



Notice that USA has highest MPG in City Mileage for Wagons

Combining the OUTPUT and DOCUMENT Destination

Up to this point, a major theme of this chapter has been this: replaying documents to the OUTPUT destination creates the potential to combine output in powerful new ways. To recap:

1. Use the REPLAY statement, sending output to the OUTPUT destination. This creates tables that you can use in the DATA step.
2. Reshape the output data using DATA step programming.
3. Create new output, possibly storing it back in your document, if desired.

This process can bring together output into a single table, even if these outputs were replayed at different times.

Contrast this process against the alternative without the Document Facility. First, you would have to search through your output to find the two tables that you want. Then you would need to re-import them into SAS data sets. Of course, you're not done yet. Since you have imported external data, you have an additional data cleaning step. Then, finally, once those chores are completed, proceed with the additional analysis.

Better yet, you could just do a replay directly from your document to a data set, keeping your data in SAS format the entire time, with minimal time and effort.

ODS OUTPUT and the DOCUMENT Procedure's Data Sets

Since the theme of this chapter is converting between ODS document objects and SAS data sets, it is worth pointing out that the output from the DOCUMENT procedure's LIST statement can itself be converted to a SAS data set via the OUTPUT destination.

This section presents two applications of this technique: Revising the list output to create a table of contents, and writing code to iterate over this list.

Simple List Output

The ODS output object for the LIST statement has the ODS name `properties`. To export a top-level list to a data set, use code such as Program 8.8.

Program 8.8: Outputting LIST Data to a Data Set

```
ods listing close;  
proc document name=mylib.firstdoc;  
ods output properties=simple; ①  
list / details;  
run; ②  
ods output close;  
  
quit;
```

- ① The ODS OUTPUT statement in this instance makes the output from this LIST statement go to the SAS data set `work.simple`. To avoid having the ODS OUTPUT destination try to capture output from other document statements, the ODS OUTPUT statement is wrapped closely around the LIST statement.
- ② Remember the ever-important RUN statement. Without this statement, you might see an error message such as this:

```
ERROR: Output destination not active
```

If you see such a message, it might come as a surprise, since the ODS OUTPUT statement did in fact, execute. However, all that the OUTPUT destination does is add output to the SAS data set at the end of each RUN group. If you omit the final RUN statement, the OUTPUT destination does not see the output from that RUN group. When the CLOSE statement is executed, the destination is no longer active and, thus, no more output is sent to it.

Contents of the Document Data Listing

Program 8.9 shows how to create a detailed listing of a document, along with BY-group variables.

Program 8.9: A Complete Listing of the Document as a Data Set

```
%let docname=mylib.firstdoc;
%let outds=detail;
proc document name=&docname.;
ods output properties=&outds.;
list / levels=all by groups details;
run;
ods output close;
quit;
proc contents data=&outds. Varnum;
run;
```

Output 8.8: The Document LIST/DETAILS Command Output to a Data Set

| Variables in Creation Order | | | | |
|-----------------------------|-----------|------|-----|-------------|
| # | Variable | Type | Len | Format |
| 1 | Path | Char | 512 | |
| 2 | Type | Char | 9 | |
| 3 | Size | Num | 8 | 12. |
| 4 | Created | Num | 8 | DATETIME19. |
| 5 | Modified | Num | 8 | DATETIME19. |
| 6 | Symlink | Char | 512 | |
| 7 | Label | Char | 256 | |
| 8 | Template | Char | 512 | |
| 9 | PageBreak | Num | 8 | |
| 10 | Sex | Char | 1 | |

The explanation for most of these was already discussed in Chapter 4, “Listing Documents Using the DOCUMENT Procedure.” You have seen `Pagebreak` in the previous chapter. `Symlink` will be discussed in Chapter 10, “Working with Links.” `SEX` is a BY-group variable. BY-group variables are last in the listing, since they vary by document.

Program 8.10: Displaying the Output Data Set

```
proc print data=detail(obs=3);
id path;
var type size created label/* full output would be very wide*/
run;
```

The first three rows of the data set are presented as Output 8.9.

Output 8.9: The Document Listing as a Data Set

| Path | Type | Size | Created | Label |
|--|-------|------|--------------------|--|
| \Univariate#1\ByGroup1#1\Height#1 \Moments#1 | Table | 619 | 14APR2012:17:47:01 | Moments |
| \Univariate#1\ByGroup1#1\Height#1 \BasicMeasures#1 | Table | 526 | 14APR2012:17:47:01 | Basic Measures of Location and Variability |
| \Univariate#1\ByGroup1#1\Height#1 \TestsForLocation#1 | Table | 578 | 14APR2012:17:47:01 | Tests For Location |

Conditionally Executing the DOCUMENT Procedure with CALL EXECUTE

You can use the listing data set, as shown in Output 8.7, to demonstrate how to execute SAS code for each element of a document listing.

You can use the CALL EXECUTE routine to write general data-driven code that has nothing to do with the DOCUMENT procedure, of course. However, the focus in this book is the convenience of using document listings as input for this technique.

The problem this program is trying to solve is that the DOCUMENT procedure provides labels automatically for some output. Although these labels accurately describe the nature of the output, they can be too terse to help you remember why you created and stored the output in the first place. For example, the document automatically assigns the label “Quantiles” for the Quantiles output from the UNIVARIATE procedure. It would be better if the label had some more detail. This program adds a few details to the object’s label based on information in the object’s path.

The technique uses the CALL EXECUTE facility to call a DOCUMENT procedure statement for each output object whose path satisfies a specific condition. You can use other variables besides the path to build condition statements. Output 8.8 contains the full list.

The CALL EXECUTE routine builds a list of code statements that are executed after the data set is complete. This list is built in First-In-First-Out (FIFO) order. When the DATA step terminates, the CALL EXECUTE routine resolves any needed macro variables and then executes the code.

The program in this example uses the data set that is generated from the LIST statement output, stored in the data set. Returning to the example at hand, notice that the procedure for this type of operation is to define a macro to perform the work, and then to arrange to have that macro called for each step of the data set loop.

Program 8.11: Building Custom Labels for All Qualifying Output Objects

```
%macro doclist(docname=>,outds=>);  
proc document name=&docname.;  
  ods output properties=&outds.;  
  list / levels=all by groups details;  
run;  
ods output close;  
quit;  
%mend;  
  
%doclist(docname=my.lib.firstdoc,outds=docdata) ①  
%macro setlbl(doc=>,obj=>,lbl=>); ②  
  proc document name=&doc. ;  
    setlabel &obj. &lbl. ;  
  quit;  
%mend setlbl;  
  
options mprint mlogic;  
data _null_;  
length command $500 ;  
set docdata;  
  
if index(path,'Quant') then do; ③  
  /* build the command */  
  /* variable is in third position on the path */  
  var=scan(path,3,'\\');  
  /* strip out sequence number */  
  pdpos=index(var,'#');  
  var=substr(var,1,pdpos-1);  
  /* build the label out of the listing components */  
  mylbl=catx('','Quantiles for ',
```

```

var': sex ='sex,
'Report year',put(today(),year4.));

command=cats("%setlb",'(doc=my lib.firstdoc,obj='
path,' ,lbl=""',my lbl,"")');
④
  put 'executing' command;
  call execute(command);
⑤
end;
run;

ods html file='newlabs.html';
proc document name=my lib.firstdoc;
  list ^(where=(_name_="Quantiles")) / details;
  run;
quit;
ods html close;

```

- ➊ The first step in writing this data-driven program is to create the driving data set. In this case, it is a listing of the document as a data set, as produced in Program 8.7. In this example, that same code is written as a macro. It is not necessary to do this step as a macro, but if you find this technique useful, then you'll find it helpful to keep in your macro library. After you run this code, the data set `docdata` will contain the listing of the document.
- ➋ Once the data set is built, creating code based on it is the next step. The code for this example writes a macro to invoke the DOCUMENT procedure, runs a command, and terminates. In this example, it is called `%setlb` for “set label”. There is no restriction on how this macro should be named. It is an ordinary SAS macro.

When the program is finally run by CALL EXECUTE, there will be one DOCUMENT procedure call for each line in the data set `detail`. If you are concerned about calling the DOCUMENT procedure multiple times, it is done this way in the interest of keeping the design of the `%setlb` macro and data set as simple as possible. In practice, the overhead of a few extra PROC DOCUMENT calls is usually acceptable.

If you use macro language statements in a CALL EXECUTE routine call, parameters for the macro can come from the global environment. As an alternative, they can come from the data set variables from the detailed document listing, as shown in Output 8.10. In this example, the macro takes as a parameter the path and a text string as the label. The text string, when passed to the macro, should include quotation marks.

- ➌ Notice that to describe the parts of the document, such as the path or creation dates, you do not need underscores. You need the underscores when these special document variables appear in the DOCUMENT procedure code, but you do not need them when they are used as DATA step variables. One advantage to using this data-driven approach is that all the familiar DATA step string functions are available. This example shows some character variable functions that are used to extract the variable used in the VAR statement from the original PROC UNIVARIATE call. Perl regular expressions would also be a possible way of testing and parsing the path.
- ➍ You must build any code that is submitted to CALL EXECUTE as a character string. This is why the invocation of `%setlb` is built as a string. This string will be sent to the CALL EXECUTE routine, and the macro will be run once for each row where the condition is true.

The value of the variable `command`, when it is finally put together, is shown in the third code block of Program 8.11.

```
%setlb(doc=mylib.firstdoc,
      path=\univariate#1\by group#1\age#1\quantiles#1,
      lbl="Quantiles for Age, Sex=F, reporting year 2011")
```

Any string that you send to CALL EXECUTE must appear exactly as it would if you were entering it in open code. This includes semicolons, if needed, at the end of the string.

- ⑤ The CALL EXECUTE routine takes a single string as an argument. The string can either be literal, or it can be stored in a variable, and the variable name is then passed. In this case, the string is the macro command from step ④.

Finally, to show that the labels were changed, the last block of code reruns the DOCUMENT procedure to obtain a new list. This technique of generating DOCUMENT procedure code to update the document is necessary, because changing the `docpath` data set does not affect the original document.

Output 8.10: The Document Relabeled. Note That the Labels Are Now Customized to Match the Path

| Listing of: \Mylib.Firstdoc\where=_name_ contains 'Quantiles') | | | | | | | | | |
|--|---|-------|---------------|--------------------|--------------------|---------------|---------------------------|---|------------|
| Order by: Insertion | | | | | | | | | |
| Number of levels: All | | | | | | | | | |
| Obs | Path | Type | Size in Bytes | Created | Modified | Symbolic Link | Template | Label | Page Break |
| 1 | \Univariate#1 \ByGroup1#1 \Height#1 \Quantiles#1 | Table | 601 | 04DEC2011:17:10:39 | 04DEC2011:17:10:39 | | base.univariate.Quantiles | Quantiles for Height : sex = F Report year 2011 | |
| 2 | \Univariate#1 \ByGroup1#1 \Age#1 \Quantiles#1 | Table | 597 | 04DEC2011:17:10:39 | 04DEC2011:17:10:39 | | base.univariate.Quantiles | Quantiles for Age : sex = F Report year 2011 | |

To summarize, the first step is to use PROC DOCUMENT and the ODS OUTPUT destination to create a data set containing the document's listing. Then, this data set is fed into a DATA step, which analyzes it, builds a command based on the listing data set, and executes that command via a macro and the CALL EXECUTE routine.

The SASEDOC Engine

In addition to the ODS OUTPUT= statement, there is another way to convert document entries to SAS data sets, and this is through the SASEDOC LIBNAME engine. An *engine* is the underlying software that lets SAS help you view a foreign data object as if it were a SAS library. Examples include the ACCESS LIBNAME engine that treats a Microsoft Access database as a library. Some engines, such as the ORACLE engine require purchase of an additional license. SASEDOC is included when you license Base SAS.

The example of interest here is the SASEDOC engine, which treats a folder in an ODS document like a SAS library. Likewise, any output objects that are stored in that folder are presented to SASEDOC as data sets in that library. Examples follow shortly.

There are a couple of warnings regarding the SASEDOC engine. First, the libraries that are created with the engine are *Read-only*. Second, the variable names might not match what you expect, since this is formatted internally for the system's convenience rather than for yours. Finally, the document pathname that is referred to in the LIBNAME statement must contain output objects at the top level of the folder used in the LIBNAME statement.

It is this author's opinion that SASEDOC should be used sparingly. The reason for this is that the ODS OUTPUT destination can extract data that is a better match to how your data is actually used in a practical problem. The ODS OUTPUT destination recognizes an object's template, a capability SASEDOC is not designed to handle.

Accessing a Directory of Output

To use the SASEDOC LIBNAME engine, you must select a folder that contains at least one output object. This is the only type of directory for which SASEDOC is suited. These output objects must be at the first level so that a simple list would capture it. *SASEDOC does not go more than one level into the directory.*

Here is the syntax for declaring a library name with the SASEDOC engine:

```
LIBNAME library-name SASEDOC absolute- path name(<options>);
```

Program 8.13 shows how to view a data set of UNIVARIATE procedure output. In this case, the variable HEIGHT for SEX=F is shown as its own SAS library.

Program 8.13: Accessing a Directory of Output

```
libname univ SASEDOC  
  '\my.lib.firstdoc\Univariate#1\By Group 1#1\Height#1';
```

The LIBNAME statement creates a libref named `univ`, pointing to the pathname `\my.lib.firstdoc\Univariate#1\By Group 1#1\Height#1`. Long pathnames like this are usually just cut and pasted from LIST statement output. The log entry below shows success.

Display 8.1: The Library is Assigned

```
|412  libname univ sasedoc '\my.lib.firstdoc\Univariate#1\By Group 1#1\Height#1';  
NOTE: Libref UNIV was successfully assigned as follows:  
  Engine:      SASEDOC  
  Physical Name: \My.lib.Firstdoc\Univariate#1\By Group 1#1\Height#1
```

Once the library is successfully assigned, you can use it like any other library as input to SAS procedures or DATA steps. For example, the DATASETS procedure looks at this directory of the document and displays the member list. The members, in this case, are the output objects stored there. The `UNIV` library can be accessed, even though in Read mode, like any other SAS library.

Program 8.14: Viewing the UNIV Library

```
proc datasets lib=univ;  
quit;
```

Output 8.11: The Results

| Directory | |
|---------------|--|
| Libref | UNIV |
| Engine | SASEDOC |
| Physical Name | \Mylib.Firstdoc\Univariate#1\ByGroup1#1\Height#1 |
| Created | 14APR12:17:47:01 |
| Last Modified | 14APR12:17:47:01 |
| Label | Height |

| # | Name | Member Type | Size | Last Modified | Label |
|---|------------------|-------------|------|------------------|--|
| 1 | BASICMEASURES | DATA | 526 | 14Apr12:17:47:00 | Basic Measures of Location and Variability |
| 2 | EXTREMOBS | DATA | 412 | 14Apr12:17:47:01 | Extreme Observations |
| 3 | MOMENTS | DATA | 619 | 14Apr12:17:47:00 | Moments |
| 4 | QUANTILES | DATA | 601 | 14Apr12:17:47:01 | Quantiles |
| 5 | TESTSFORLOCATION | DATA | 578 | 14Apr12:17:47:01 | Tests For Location |

If the libref `mylib` is not defined, or if the path is not valid for any other reason such as a misspelled path element or incorrect sequence number, you would see a failure error message, saying there is an error in the LIBNAME statement. The error message does not tell you what is malformed.

Program 8.15: Accessing ODS Output via the SASEDOC LIBNAME Engine

```
title 'Using the SASEDOC Engine';
proc print data=univ.moments;
run;
ods html close;
```

For example, the PRINT procedure accesses the document object as if it were the data set `univ.moments`.

The output of the PRINT procedure is shown in Output 8.12.

Output 8.12: The Data Set UNIV.Moments

| Obs | Label1 | cValue1 | nValue1 | Label2 | cValue2 | nValue2 |
|-----|-----------------|------------|-----------|------------------|------------|---------|
| 1 | N | 9 | 9.000 | Sum Weights | 9 | 9.000 |
| 2 | Mean | 60.5888889 | 60.589 | Sum Observations | 545.3 | 545.300 |
| 3 | Std Deviation | 5.01832752 | 5.018 | Variance | 25.1836111 | 25.184 |
| 4 | Skewness | -0.7238643 | -0.724 | Kurtosis | -0.3464949 | -0.346 |
| 5 | Uncorrected SS | 33240.59 | 33240.590 | Corrected SS | 201.468889 | 201.469 |
| 6 | Coeff Variation | 8.28258714 | 8.283 | Std Error Mean | 1.67277584 | 1.673 |

As with the OUTPUT destination, variables such as `LABEL1` or `nVALUE1` are named based on the output template that SAS plans to use. The names are not necessarily chosen to be in harmony with how you prefer to name and arrange your data. Always double check the names of the variables that were created either through ODS output data sets or through SASEDOC.

One disadvantage of the SASEDOC engine is that it might lack the context that you need.

Applications of the SASEDOC Engine

SASEDOC works well when you want to see inside a document, particularly graphics that you saved with the DEVICE=JAVAIMG, JAVA, ACTXIMG, or ACTIVEX options.

First, let's look at how Program 8.16 creates the graph and saves it in a document.

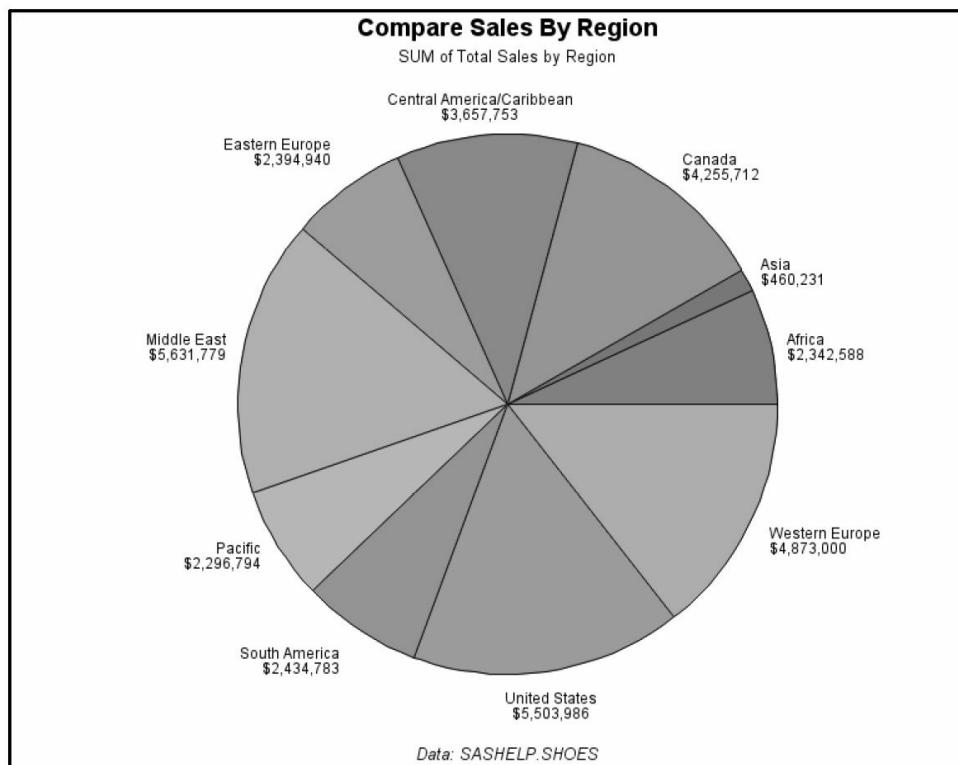
Program 8.16: Saving and Recalling a Graphics Object as a Table

```
footnote1 "data: sashelp.shoes";
title1 "Compare Sales By Region";
goptions device=actximg;
ods document name=oldgraphtest(write);

proc gchart data=sashelp.shoes;
  pie region / sumvar=sales;
run;
quit;
ods document close;
```

The graph is shown (shaded in gray) as Output 8.13.

Output 8.13: The Original Graph



To determine the path reference for the location of the pie chart within the document `work.oldgraphtest`, it is necessary to display a listing of the document. Assuming this is done, the next step is to create a library with the SASEDOC engine. The pathname must be an absolute pathname, including the document name through the path that contains the output object. Program 8.17 creates the library and verifies that it contains a data set by calling the DATASETS procedure.

Program 8.17: Viewing the Document as a SAS Library

```
libname oldgraph sasedoc "\work.oldgraphtest\gchart";
ods html file="program8_17.html";
proc datasets lib=oldgraph;
quit;
ods html close;
```

Output 8.14 shows that the library contains a single data set, named `gchart`.

Output 8.14: \WORK.OLDGRAPHTEST\GCHART As a Library

Compare Sales By Region

| Directory | |
|---------------|-----------------------------|
| Libref | OLDGRAPH |
| Engine | SASEDOC |
| Physical Name | \Work.Oldgraphtest\Gchart#1 |
| Created | 27APR12:12:26:15 |
| Last Modified | 27APR12:12:26:15 |
| Label | The Gchart Procedure |

| # | Name | Member Type | Size | Last Modified | Label |
|---|--------|-------------|------|------------------|---------------------|
| 1 | GCHART | DATA | 950 | 27Apr12:12:26:15 | Pie chart of Region |

Data: SASHELP.SHOES

Now, you can use the OLDGRAPH.GCHART data set. Just remember that it is Read-only. Program 8.18 shows how it is rebuilt as a table. Only the data necessary for building this one graph is contained in the data set.

Program 8.18: Printing the Chart in Tabular Form

```
ods html file="program8_18.html";
proc print data=oldgraph.gchart;
format __Freq__ 5.; /* these are double underscores */
var region sales __freq__ / style={font_size=5};
run;
ods html close;
```

Output 8.15: Graph Returned to Tabular Format

| Compare Sales By Region | | | |
|-------------------------|---------------------------|-------------|----------|
| Obs | Region | Sales | __FREQ__ |
| 1 | Africa | \$2,342,588 | 56 |
| 2 | Asia | \$460,231 | 14 |
| 3 | Canada | \$4,255,712 | 37 |
| 4 | Central America/Caribbean | \$3,657,753 | 32 |
| 5 | Eastern Europe | \$2,394,940 | 31 |
| 6 | Middle East | \$5,631,779 | 24 |
| 7 | Pacific | \$2,296,794 | 45 |
| 8 | South America | \$2,434,783 | 54 |
| 9 | United States | \$5,503,986 | 40 |
| 10 | Western Europe | \$4,873,000 | 62 |

Data: SASHELP.SHOES

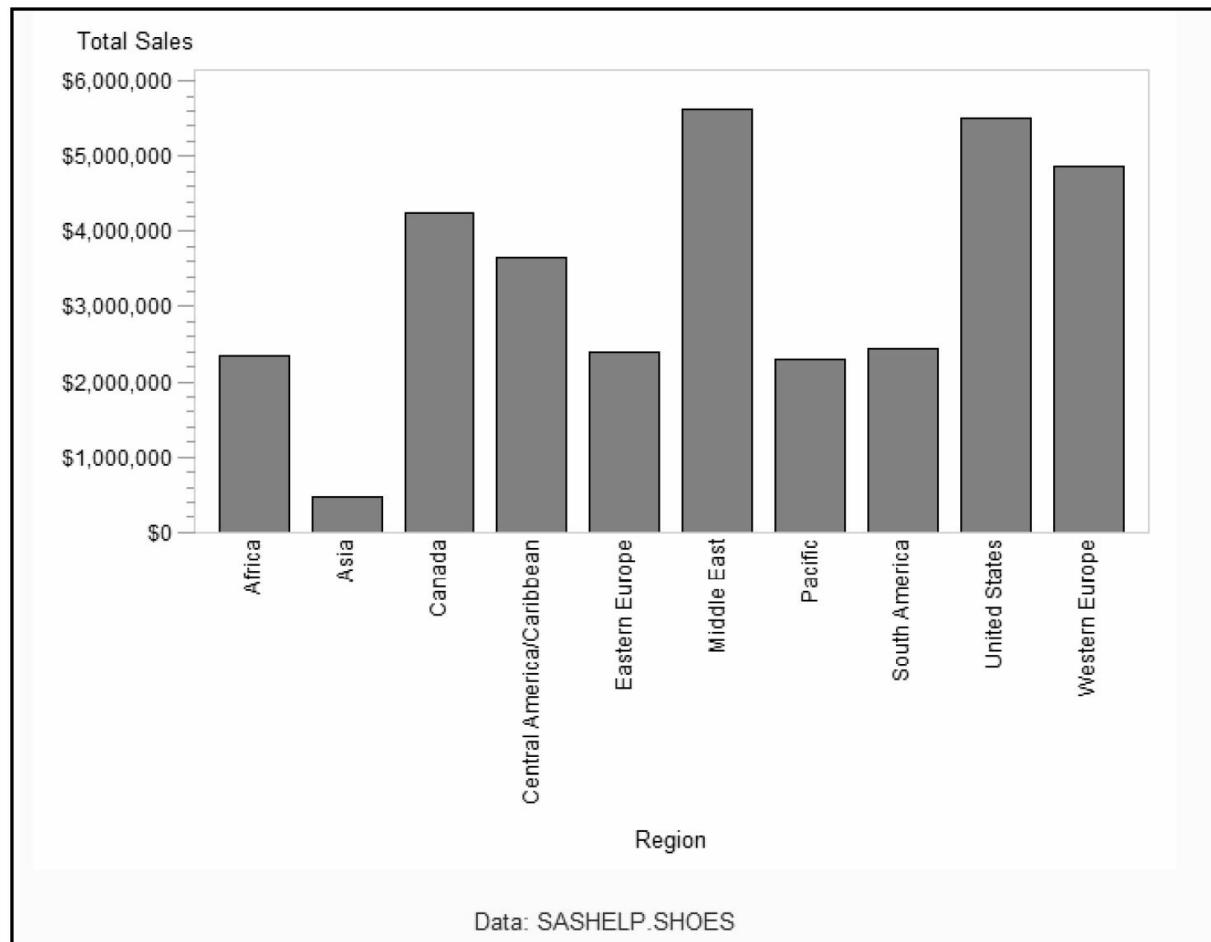
With the data restored to tabular format, you can rebuild this graph with different parameters and options without having to run the original analysis.

Program 8.19 shows the same data rendered as a vertical bar chart. The point is that you can now continue to experiment with options until you get your output precisely as you want it, and again, without having access to the original analysis.

Program 8.19: Re-Rendering a Graph

```
ods html file="program8_19.html";
proc gchart data=oldgraph.gchart;
vbar region / sumvar=sales;
run;
quit;
ods html fclose;
```

Output 8.16: A Different Representation of the Same Data



The SASEDOC LIBNAME engine gives you many of the same abilities that you have with the ODS OUTPUT statement to create data sets from output that is stored in the document. The one disadvantage is that you have to set up a separate library for each output directory from which you need data.

Program 8.20: An Empty Library Because There Are No Output Objects

```
proc datasets lib=nothing;
quit;
```

Output 8.17: Empty Library Warning

```
740 libname nothing sasedoc '\mylib.quickstart\univariate';
NOTE: Libref NOTHING was successfully assigned as follows:
  Engine:      SASEDOC
  Physical Name: \Mylib.Quickstart\Univariate#1
741 proc datasets lib=nothing;
          Directory
          Libref      NOTHING
          Engine     SASEDOC
          Physical Name \Mylib.Quickstart\Univariate#1
          Created    27APR12:10:15:40
          Last Modified 27APR12:10:15:45
          Label      The Univariate Procedure
WARNING: No matching members in directory.
742 quit;
```

The directory in question contains several subfolders, one for each BY-group output stored. Even though you would see output objects if you went all the way down to the appropriate level, there

are none at the current level. Hence, there are no data sets in the library.

The SASEDOC Engine and Sequence Numbers

The SASEDOC engine creates views of the data sets in a document path, but includes only the most recent sequence number. If you need to see an earlier sequence number, use the `DOC_SEQNO=` option. Examples of the `DOC_SEQNO` will be covered shortly.

It is a bit tricky to create a situation where you would require `DOC_SEQNO`. Typically, if you rely exclusively on folders that were created by SAS procedures, you would rarely encounter a situation where you would be required to use `DOC_SEQNO`.

Program 8.7 showed an example of how a folder can contain several output objects with the same name, necessitating multiple sequence numbers. The path `new_exhibits\sgpanel` contains three graphs named `sgpanel`. The `DOC_SEQNO` option is necessary to see all of them.

Output 8.18: The Graph Example from SASHELP.CARS

| Listing of: \Mylib.Quickstart\new_exhibits#1\Sgpanel#1 | | |
|--|-----------|-------|
| Order by: Insertion | | |
| Number of levels: 1 | | |
| Obs | Path | Type |
| 1 | SGPanel#1 | Graph |
| 2 | SGPanel#2 | Graph |
| 3 | SGPanel#3 | Graph |

Program 8.21 creates the SASEDOC library corresponding to this directory.

Program 8.21: Viewing a Directory with Only One ODS Name

```
libname graphme sasedoc 'mylib.quickstart\new_exhibits\sgpanel';
proc datasets lib=graphme;
quit;

ods html close;
```

In Output 8.19 there is only one data set named `SGPANEL`, even though there are three objects in the directory. The label for the data set `SGPANEL` shows that this is the graph for `VEHICLE_TYPE=WAGON`.

Output 8.19: SASEDOC View of GRAPHME Library

| Directory | | | | | |
|---------------|--|--|--|--|--|
| Libref | GRAPHME | | | | |
| Engine | SASEDOC | | | | |
| Physical Name | \Mylib.Quickstart\new_exhibits#1\Sgpanel#1 | | | | |
| Created | 27APR12:11:31:08 | | | | |
| Last Modified | 27APR12:11:31:09 | | | | |
| Label | The Sgpanel Procedure | | | | |

| # | Name | Member Type | Size | Last Modified | Label |
|---|---------|-------------|------|------------------|--------|
| 1 | SGPANEL | DATA | 3100 | 27Apr12:11:31:08 | Wagons |

You can use the DOC_SEQNO option in any command that takes data set options in order to see the data sets corresponding to other versions. Program 8.22 shows an example with the PRINT procedure. However, in this case, all three graphs are actually stored internally as the same table. Program 8.22 accesses the sgpanel#2 version of the output object sgpanel.

Program 8.22: Accessing Other Versions of the SGPANEL Table

```
ods html file="program8_22.html";
proc print data=graphme.sgppanel(doc_seqno=2);
run;

ods html close;
```

Output 8.20 shows that sometimes there is more in the data set than is needed to replay the output. In the case of this example, any one of the SGPANEL tables will suffice.

Output 8.20: The Table Contains Enough Data for All Three Graphs

| | | | | |
|----|--------|---------|--------|------|
| 17 | Sports | Highway | Europe | 26.0 |
| 18 | Sports | Highway | USA | 25.0 |
| 19 | Truck | City | Asia | 16.5 |
| 20 | Truck | City | USA | 15.5 |
| 21 | Truck | Highway | Asia | 19.5 |
| 22 | Truck | Highway | USA | 19.5 |
| 23 | Wagon | City | Asia | 21.0 |
| 24 | Wagon | City | Europe | 19.0 |
| 25 | Wagon | City | USA | 22.0 |
| 26 | Wagon | Highway | Asia | 28.0 |
| 27 | Wagon | Highway | Europe | 26.5 |
| 28 | Wagon | Highway | USA | 30.0 |

The SQL procedure provides a common case where DOC_SEQNO would be necessary to see all of the items in a directory. Program 8.23 shows that multiple sequence numbers can arise when there are several queries in a single SQL procedure call.

Program 8.23: A Document Where DOC_SEQNO Is Necessary

```

ods document name=mylib.queries(write);
proc sql;
  title 'Result 1: USA';
  select *
  from sashelp.cars
  where origin='USA';

  title 'Result 2: MPG by Origin';
  select origin,mean(mpg_highway)
  from sashelp.cars
  group by origin;

  title 'Result 3: High Mileage Cars';
  select *
  from sashelp.cars
  where mpg_highway > 25;
quit;
ods document close;

proc document name=mylib.queries;
list sql;
run;
quit;

```

Output 8.21: A Document Where DOC_SEQNO Is Necessary

Result 3: High Mileage Cars

| Listing of: \Mylib.Queries\SQL#1 | |
|----------------------------------|-------|
| Order by: Insertion | |
| Number of levels: 1 | |
| Path | Type |
| SQL_Results#1 | Table |
| SQL_Results#2 | Table |
| SQL_Results#3 | Table |

Program 8.24 sets up the SASEDOC engine and prints out the second query.

Program 8.24: Three Tables Printed with the DOC_SEQNO Option

```
libname s sasedoc 'my.lib.queries\sql';
title 'Second Table';
proc print data=s.sql_results(doc_seqno=2);
run;
```

The output is below.

Output 8.22: Printing the Second Query with the DOC_SEQNO Option

| Second Table | | |
|--------------|--------|----------|
| Obs | Origin | mean_mpg |
| 1 | Asia | 28.266 |
| 2 | Europe | 26.008 |
| 3 | USA | 26.014 |

When you use the OUTPUT destination, it is often much easier to gather output and still maintain information about the origin of the output. For example, you already saw that if you used BY-groups, the resulting tables sent to the OUTPUT destination will include the BY-group variables. This makes it easier to conduct further analyses, since the BY-groups give you some idea of the origin and meaning of each statistic you report. Program 8.25 shows another example of this useful feature.

The OUTPUT destination is capable of creating a single table with less code, and it is easier to manage one table than many small ones. The author recommends the OUTPUT destination over the SASEDOC LIBNAME engine, but the information about SASEDOC is presented for the sake of completeness.

Program 8.25: Returning to the OUTPUT Destination

```
ods output parameterestimates=pe;
proc document name=my.lib.firstdoc;
replay / dest=output;
```

```

run;
quit;
proc print data=pe;
  run;
ods output close;

```

If you prefer several tables, you can always use DATA step code to break up a table. See the *SAS ODS User's Guide* (especially the “ODS OUTPUT Statement” and the “Dictionary of Language Statements”) for information about how to reconfigure the OUTPUT destination to produce multiple data sets instead of a single large one. In Output 8.23, the missing value of Sex corresponds to the analysis without BY-groups, which was stored at the absolute pathname `\mylib.firstdoc\REG#2`.

Output 8.23: Combining REG Tables via Output Destination

| The OUTPUT destination saves BY variable information | | | | | | | |
|--|-----|--------|-----------|-----------|----|------------|----------|
| Obs | Sex | Model | Dependent | Variable | DF | Estimate | StdErr |
| 1 | F | MODEL1 | Weight | Intercept | 1 | -117.36980 | 41.04734 |
| 2 | F | MODEL1 | Weight | Height | 1 | 3.42441 | 0.67542 |
| 3 | M | MODEL1 | Weight | Intercept | 1 | -141.10102 | 54.91776 |
| 4 | M | MODEL1 | Weight | Height | 1 | 3.91255 | 0.85700 |
| 5 | | MODEL1 | Weight | Intercept | 1 | -143.02692 | 32.27459 |
| 6 | | MODEL1 | Weight | Height | 1 | 3.89903 | 0.51609 |

Summary

Both the ODS OUTPUT statement and SASEDOC engine provide ways to access data in an item store as conventional data sets.

The ODS OUTPUT statement exports data directly into a new data set, which can then be used as any other data set. The ODS OUTPUT statement is capable of aggregating all output of a given type to the same data set. This capability permits new insight by making comparisons easier between results that are stored in different places in the item store.

The SASEDOC engine provides an alternate method to create a data set from an output object in the item store. The SASEDOC engine is particularly useful for rebuilding graphs, but only when those graphs are saved with specific graphics devices.

Chapter 9: Importing Data to the Document

[Overview](#)

[The Document Is Not a SAS Library](#)

[The IMPORT Statement](#)

[Data for Examples](#)

[Importing a Data Set](#)

[Importing Text Files](#)

[Importing Data and Its Template to a Document](#)

[Summary](#)

Overview

In the previous chapter, you saw how to export SAS procedure output that is stored in documents to SAS data sets. In this chapter, you will learn how to import SAS data sets into a document as output objects. You will also learn how to import Graphic Segment Catalogs (GRSEGs). Finally you will learn how to import a text file directly to a document. This feature is new for SAS 9.3. This makes it convenient to store each program log along with its output. When you store your log along with your output, you make it easier for others to review and understand your work.

There are other cases where importing might be useful: In the course of writing a report, you might need, to cite the source of your data. It might be useful to start your report by printing a small summary table, as part of the introduction to your report. By importing a data set into the directory where related exhibits are stored, you can easily replay the table, along with other supporting exhibits in the same folder.

See the appendix in this book for information on importing GRSEG entries.

The Document Is Not a SAS Library

When you consider whether to import a data set into a document, keep in mind the purpose of the ODS document: To replay related illustrations together in order to make an effective point (without rerunning the original analysis, of course). Thus, it is unwise to store large tables in the document, even if there were no performance considerations, since doing so may result in voluminous files when you replay the output. Remember: The document is not intended as a substitute for a SAS library. Furthermore, the document does not store the original data set name in the document, although you can always label the imported data set manually with a SETLABEL command.

The IMPORT Statement

Here is the syntax for the IMPORT statement:

```
IMPORT DATA=data-set-name | GRSEG = Graphics segment entry | TEXTFILE= TO path/positioning argument;
```

You must use DATA=, GRSEG=, or TEXTFILE= to import the appropriate object. Importing GRSEG entries will be covered in the appendix.

Data for Examples

As the first example of importing data sets into a document, consider the case of a small reference table that needs to be included in your report. Program 9.1 shows how you can make project documentation easier by storing the names of data sets that were used in your project. When people replay the document, they can see this table along with the other output that you replay. Such a table would be helpful in orienting others to the key data sets used to create your analyses. Other examples of useful things to store in a document might be a table explaining the meaning of codes that you used.

Program 9.1: A Reference Table for Importing into a Document

```
data setsused;
label dsname = 'Data Set Name' comment = 'Description';
length dsname $31 comment $60;
infile datalines truncover;
input dsname &$30. comment &$60.:
datalines;
sashelp.fitness Fitness Data
sashelp.snacks Promotion Effectiveness Data for Snacks
sashelp.classfit Fitness Data for the SASHELP.CLASS folks!
;
run;

ods html style=htmlblue;
title 'Data Sets Used for This Project';
proc print data=setsused noobs;
run;
```

Output 9.1: A Small Reference Table

Data Sets Used for This Project

| dsname | comment |
|------------------|---|
| sashelp.fitness | Fitness Data |
| sashelp.snacks | Promotion Effectiveness Data for Snacks |
| sashelp.classfit | Fitness Data for the SASHELP.CLASS folks! |
| sashelp.cars | Mileage Data for Cars |

Importing a Data Set

Program 9.2 imports a data set into the document MYLIB.CARS as presented in Chapter 1 “Introduction and Quick-Start Guide,” inserting the data set at the default position at the end of the document.

Program 9.2: Importing a Data Set into MYLIB.CARS

```
ods html file='program9_2.html';
proc document name=mylib.quickstart;
make reference_tables;
setlabel reference_tables "Useful lookup table";
import DATA=setsused to tblsused; ①
run;

setlabel reference "Exhibit 1: Tables used in Analysis";
title;
list / details;
run;
quit;
ods html close;
```

- ① Every DOCUMENT statement must include a path argument, which in this case is `tblsused1`. It is critical to emphasize that there is no default for providing a path, and that it is not the same as the imported filename or SAS file handle. Since no positioning argument has been given, the data set is inserted at the last place in the current folder. Because the order matters when replaying a document, you must specify a position argument (or accept the default, which is to put it at the end of a folder).

It is helpful to choose a name for your path that is easy to remember and easy to type, but it does not have to be the same as the data set name. Keep in mind that you can use the SETLABEL statement to provide additional documentation about the nature of the tables that you import.

The output from Program 9.2 shows the listing of the document with the imported table sets that are used. Once the data is imported, the document views it as it does any other output object of type table. The document, for practical purposes, does not care that this table was once a SAS data set. You can now do anything to an imported table that you can do to any other output object.

Output 9.2: Document with Imported Table Sets

| Listing of: \Mylib.Quickstart | | | | |
|-------------------------------|---------------|-------|------------------|------------------------------------|
| Order by: Insertion | | | | |
| Number of levels: 1 | | | | |
| Obs | Path | Type | Size in Bytes | Label |
| 1 | \Contents#1 | Dir | | The Contents Procedure |
| 2 | \Univariate#1 | Dir | | The Univariate Procedure |
| 3 | \Means#1 | Dir | | The Means Procedure |
| 4 | \reference#1 | Table | 344 | Exhibit 1: Tables used in Analysis |

Program 9.3: Viewing the Table

```
ods html file='program9_3.html';
proc document name=mylib.quickstart;
replay reference_tables\tblsused; ①
run;
quit;
ods html close;
```

- ① The result, shown in Output 9.3, is nearly identical to the original table, but there is one difference. Even though the data set was printed with the NOOBS option to repress observation numbers, they are nonetheless here. This is not a bug. Why does the column OBS appear here? The data set is maintained in the document with the template BASE.DOCUMENT.TABLE, and this template includes a column for the observation number (obs). Getting rid of these observation numbers requires knowing how to modify a template, which is covered in Chapter 11, “Working with Templates.”

Importing Text Files

As of SAS 9.3, you can import text files directly into the document. In addition, you can print them out with additional context such as titles. They are printed out using a special mode that appears somewhat like a SAS LISTING output, using a monospace font.

As an example, Program 9.4 shows how SAS log files can be saved to external files via the PRINTTO procedure, and then imported into the document. One possible application, outlined in the examples that follow, shows how the log that produced some output is stored in the document near that output. This program assumes that the SAT scores data set has already been read into the WORK library. The DATA step code to accomplish this is on the author page for this book as well as being available in the *Step-by-Step Programming with Base SAS Software*.

Since this demonstration imports a SAS log, Program 9.4 runs some sample analyses to create a log and a document. The PRINTTO procedure reroutes log files to an external file.

Program 9.4: Creating a Log and a Document

```
filename mylog='program9_4.log'; /* point to log file */
proc printto log=mylog/* reroute log to file */
run;

ods document name=mylib.sat(write) dir=(path=\summaries label='Routine Summaries');
/* some summary data/checking code */
proc means data=sat_scores mean nmiss max min;
class gender year test;
var satScore;
run;
proc freq data=sat_scores;
table gender year test;
run;
ods document close;
proc printto;
run;
filename mylog clear; /* deassign file to make sure its unlocked */
```

Display 9.1. is a log showing that the PRINTTO statement has finished in Program 9.4, so the log has been stored in the document.

Display 9.1: Exporting a Log to a File

```
388 filename mylog 'program9_5.log'; /* point to log file */
389 proc printto log=mylog/* reroute log to file */
390 run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time          0.00 seconds
      cpu time          0.00 seconds

404 filename mylog clear;
NOTE: Fileref MYLOG has been deallocated.
```

Now that the log is available as a file, you can import it to your document. Program 9.5 captures the log. The line NEWFILE=NONE in the ODS HTML statement keeps all output on the same page.

Program 9.5: Capturing the Log

```
ods html file='program9_5.html' newfile=none;
proc document name=mylib.sat;
make logs;
```

```

import textfile=mylog to logs\example;
run;
list / levels=all;
run;
replay logs;
run;
quit;
ods html close;

```

Output 9.3: The Text File as It Appears in the Document Listing and Printed Out

| Listing of: \Mylib.Sat\ | | |
|-------------------------|-------------------|-------|
| Order by: Insertion | | |
| Number of levels: All | | |
| Obs | Path | Type |
| 1 | \summaries#1 | Dir |
| 2 | \logs#1 | Dir |
| 3 | \logs#1\example#1 | Batch |

```

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time          0.00 seconds
      cpu time          0.01 seconds

462
463 ods document name=mylib.sat(write) dir=(path=\summaries label='Routine Summaries');
464 /* some summary data/checking code */
465 proc means data=sat_scores mean nmiss max min;
466 class gender year test;
467 var satScore;
468 run;

NOTE: There were 108 observations read from the data set WORK.SAT_SCORES.
NOTE: PROCEDURE MEANS used (Total process time):
      real time          0.03 seconds
      cpu time          0.03 seconds

469 proc freq data=sat_scores;
470 table gender year test;
471 run;

NOTE: There were 108 observations read from the data set WORK.SAT_SCORES.
NOTE: PROCEDURE FREQ used (Total process time):
      real time          0.05 seconds
      cpu time          0.01 seconds

472 ods document close;
473 proc printto;

```

Notice that the imported log is a type of object that we have not encountered before, a batch object. The batch type is just another type of output object. In fact, you can add titles and footnotes as well as object notes to it before reprinting, as shown in Program 9.6.

Being able to store your logs with your code enhances your ability to document your work. Even if the analysis data set is no longer available, the more logs you save to the document, the easier it is to remember the precise conditions under which your code was run.

Program 9.6: Adding Some Documentation to a Log

```
ods html file='program9_6.html';
proc document name=mylib.sat;
dir logs;

obtitle example 'This is the log for the SAT Scores analysis, run on
                  April 30, 2012';
replay;
run;
quit;
ods html close;
```

Output 9.4: A Log with a Title

This is the log for the SAT Scores analysis, run on April 30, 2012

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time          0.00 seconds
      cpu time          0.01 seconds

462
463 ods document name=mylib.sat(write) dir=(path=\summaries label='Routine Summaries');
464 /* some summary data/checking code */
465 proc means data=sat_scores mean nmiss max min;
466 class gender year test;
```

Importing Data and Its Template to a Document

You might want to import a table and instruct the system to use a specific template when replaying it. In SAS 9.3, the IMPORT statement does not provide an option to import a table with a specific template. A discussion of how to do this is included in Chapter 11, “Working with Templates.”

Summary

A feature that is new for SAS 9.3 is the ability to import text files directly into documents. A useful application of this is the ability to store log files in your document. Although the example used log files, which are an important part of the project documentation process, you can import any text file. Imported tables are considered as output objects (with type=TABLE). You can give an imported table some context, such as titles and footnotes, subtitles, before- and after-notes. This makes it easy to replay your table along with some text.

You use the IMPORT statement to import small data sets, which can be replayed directly. Although there is no formal limit, the design idea was to permit replaying of illustrative tables rather than substitute for a SAS library.

Chapter 10: Working with Links

[Introduction](#)

[Creating a Link](#)

[Source and Target](#)

[Replaying Links](#)

[The FOLLOW Option of the LIST Statement](#)

[Orphaned Links](#)

[Document Management Using Links](#)

[Modifying PDF Bookmarks Using Links](#)

[Hard Links](#)

[Summary](#)

Introduction

In Chapter 6, “Managing Folders,” you saw how to create custom reports from your SAS procedure output by copying a collection of related output objects, notes, and subfolders required for the report to their own folder.

Although copying is a fine approach to take, sometimes you do not want to create a second copy of an object just to be able to store output in a different order. Links combine many of the advantages of copying, but without requiring you to manage multiple versions of the same output.

After defining the concept of a link, the chapter starts with the use of the LINK TO statement, and includes a number of examples that demonstrate the utility of creating folders or documents that consist entirely of links.

The chapter closes with a discussion of hard links, which combine the ability to create shortcuts with the ability to actually modify the linked object via the shortcut.

Creating a Link

Let's jump right in and create a link. The first type of link, and the one that you will probably use the most, is a symbolic link, or *soft link*. Either of these synonymous terms is used to define an indirect reference to an object. An indirect reference refers to the location of an object substituted for the object itself. This object can be elsewhere in the same document as the link, or in another document. You likely already have some experience with links; shortcuts to applications on both Windows and Macintosh operating systems work similarly to links. Links in documents are aliases for something that is stored elsewhere.

It is important to consistently use the terminology of links: When you link to another object, the data being linked is called the *source*. The link itself is called the *target*.

Not surprisingly, you create links with the LINK TO statement.

Syntax of the LINK TO Statement

```
LINK <source-object> TO pathname / <HARD> <LABEL> <positioning-argument>
```

Note the similarities to the COPY TO statement. With the exception of the HARD option, which is discussed later, the two look and work similarly. Program 10.1 creates a link to a folder in MYLIB.FIRSTDOC. In this example, the link itself is stored in MYLIB.FIRSTDOC, although there is no requirement that the source data be in the same document as the link.

Program 10.1: Creating a Symbolic Link

```
ods html file='program10_1.html' style=defaultbigger;
proc document name=mylib.firstdoc;
make link_folder;
dir link_folder;
run;
link \reg#1\bygroup 1\model1\fit1\obswisestats\weight to firstlink;
list / details;
run;
quit;
ods html close;
```

The listing, shown as Output 10.1, shows that the path \link_folder\firstlink is now a shortcut to

\reg#1\bygroup 1\model1\fit1\obswisestats\weight.

Output 10.1: Listing Showing the Link

| Listing of: \Mylib.Firstdoc\link_folder#1 | | | | | | | |
|---|--------------------------------|------|---------------------|---|----------|-------|---------------|
| Order by: Insertion | | | | | | | |
| Number of levels: 1 | | | | | | | |
| Obs | Path | Type | Size in Bytes | Symbolic Link | Template | Label | Page Break |
| 1 | \link_folder#1 \firstlink#1 | Link | 55 | \Reg#1\Bygroup1\Model1\Fit1 \Obswisestats\Weight | | | |

Source and Target

The results of Program 10.1 can be used to illustrate important terms. Links create a correspondence between a *source* data set and a *target link*. `Firstlink#1` is the target in this example, and the folder `\reg#1\bygroup1\model1\fit1\obswisestats\weight` is the *source data*.

To help you remember which is which, remember that the LINK TO statement has the same structure as a COPY TO statement. Recall the basic syntax of the COPY TO statement from Chapter 6, “Managing Folders.”

```
COPY source-data TO target-path;
```

With the LINK TO statement, the source plays the same role as it does in the COPY statement. The *target* link still plays a similar role to that in the COPY TO statement. However, instead of holding a copy of the source, it holds only a record of its path. You will see more specific examples of how the terms *source* and *target* are used in the following programs.

Replaying Links

Once you create a link, you can replay it as you would replay any other object, as Program 10.2 shows.

Program 10.2: Replaying Links

```
ods html file='program10_2.html' style=default;
proc document name=mylib.firstdoc;
title "Replaying the Link";
dir link_folder;
replay firstlink#1 / levels=all activetitle;

run;
quit;
ods html close;
```

Output 10.2 shows the replayed folder. You can use soft links to give other users a read-only view of your output. The source data in a soft link cannot be modified via the link.

Output 10.2: Replaying a Link

| Sex=F | | | | | |
|-----------------------------|----|--------------------|----------------|---------|---------|
| Number of Observations Read | | | | | 9 |
| Number of Observations Used | | | | | 9 |
| Analysis of Variance | | | | | |
| Source | DF | Sum of Squares | Mean Square | F Value | Pr > F |
| Model | 1 | 2362.53520 | 2362.53520 | 25.71 | 0.0014 |
| Error | 7 | 643.35369 | 91.90767 | | |
| Corrected Total | 8 | 3005.88889 | | | |
| Root MSE | | 9.58685 | R-Square | 0.7860 | |
| Dependent Mean | | 90.11111 | Adj R-Sq | 0.7554 | |
| Coeff Var | | 10.63892 | | | |
| Parameter Estimates | | | | | |
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > t |
| Intercept | 1 | -117.36980 | 41.04734 | -2.86 | 0.0244 |
| Height | 1 | 3.42441 | 0.67542 | 5.07 | 0.0014 |

The FOLLOW Option of the LIST Statement

Output 10.1 showed how links are typically displayed in a list. However, you also have the option of seeing the actual source document to which the link points. The LIST statement, covered in Chapter 4, “Listing Documents Using the DOCUMENT Procedure,” has an option that was not discussed in that chapter. In the LIST statement, the FOLLOW option creates a listing by following all links, printing the source’s folder structure as if it were part of the current document. This can be helpful for understanding what will be replayed when the link is replayed. Program 10.3 shows the FOLLOW option.

Program 10.3: Following a Link

```
ods html file="program10_3.html";
proc document name=mylib.softly;
list / levels=all follow;
run;
quit;
ods document close;
```

Output 10.3 shows that the contents of the folder `univariate#1\bygroup4#1` are displayed as though they were stored in the directory `linkdir#1`.

Output 10.3: The LIST Statement with the FOLLOW Option

Listing of: \Mylib.Softly

Order by: Insertion

Number of levels: All

| Obs | Path | Type |
|-----|--|-------|
| 1 | \linkdir#1 | Dir |
| 2 | \Univariate#1\ByGroup4#1 | Dir |
| 3 | \Univariate#1\ByGroup4#1\MSRP#1 | Dir |
| 4 | \Univariate#1\ByGroup4#1\MSRP#1\Moments#1 | Table |
| 5 | \Univariate#1\ByGroup4#1\MSRP#1\BasicMeasures#1 | Table |
| 6 | \Univariate#1\ByGroup4#1\MSRP#1\TestsForLocation#1 | Table |
| 7 | \Univariate#1\ByGroup4#1\MSRP#1\Quantiles#1 | Table |
| 8 | \Univariate#1\ByGroup4#1\MSRP#1\ExtremeObs#1 | Table |

Orphaned Links

The FOLLOW option is a reminder that there must be a valid source to follow. If you create a bad link, ODS will not complain. By design, ODS does not attempt to validate soft links when they are created.

Program 10.4 and Display 10.1 show a bad link that has been created. The LIST statement itself runs without error. When you replay the bad link is flagged with a warning.

Program 10.4: It Is Possible to Create a Soft Link to Nowhere

```
ods listing;
proc document name=badlink;
  make linkdir; /* not necessary but helpful. If problems exist, they are isolated to one folder */
  dir linkdir;
  link \mylib.firstdob\univariate#1 to badlink;
  list / details;
  run;
quit;
```

Display 10.1: A Bad Link and the Warning Message

```
1193 ods listing;
1194 proc document name=badlink;
1195   make linkdir; /* not necessary but helpful. If problems exist, they are isolated to one
1195! folder */
1196   dir linkdir;
1197   link \mylib.firstdob\univariate#1 to badlink;
1198   list / details;
1199   replay;
1200   run;

WARNING: Invalid document path: \Work.BadLink\linkdir#3\BadLink#1.
1201 quit;
```

Listing of: \Work.BadLink\linkdir#4
Order by: Insertion
Number of levels: 1

| Obs | Path | Type | Size in Bytes | Symbolic Link |
|-----|----------------------|------|------------------|------------------------------|
| 1 | \linkdir#4\badlink#1 | Link | 21 | \MYLIB\FIRSTDOB\Univariate#1 |

Checking the integrity of the links is your responsibility. If you were to create a document in which \MYLIB.FIRSTDOB\UNIVARIATE#1 were a valid path, this program would subsequently replay without error.

Document Management Using Links

In Chapter 6, “Managing Folders,” you saw that reorganizing output objects into a flatter folder structure changes the appearance of the bookmark frame that accompanies both PDF and HTML output. In this book, PDF bookmarks are covered, but the exposition works equally well for HTML files. You can find more information about the HTML bookmark file in *Output Delivery System: The Basics and Beyond*, Chapter 3, “HTML Output” in the section entitled “Generating a Table of Contents.”

The examples from Chapter 6, which utilized the COPY TO statement, can be done equally well with the LINK TO statement. And with the LINK TO statement your document does not increase much in size, compared with copying every output object that you want to print.

One way to arrange output without disturbing the original document is to make a second document consisting entirely of links to the first. Program 10.5 illustrates this process by creating links to four output objects in the `mylib.quickstart` document. Of course, you can create links to folders as well as to output objects.

Program 10.5: Document of Links

```
ods html file="program10_5.html"
title "A Directory of Links as a Starting Point for a Report on Trucks";

proc document name=softly(write);
list \mylib.quickstart\univariate#1 / levels=1 details;
/* to get a list of what the BY-groups mean */ ①
*-----;
make linkdir;
setlabel linkdir "Truck Comparison"; ②
*-----;
dir linkdir;
link \mylib.quickstart\Univariate#1\ByGroup4#1 to asia_truck/ first label; ③
*-----;
link \mylib.quickstart\Univariate#1\ByGroup13#1 to usa_truck; ④
*-----;
run;
dir ^^; ⑤
run;
quit;
```

To assist with identifying which `bygroup<n>` folder stores which by-group output, the first line ① of the program produces a listing of the document for reference. Next, ② the program creates a folder to hold the links, and labels it with a reader-friendly label. It is generally good practice to label any folder that you create. This makes your document a readable journal of your output.

The folder `linkdir#1` will hold shortcuts to two sets of SAS procedure output: the first for `ORIGIN=ASIA` and the second for `ORIGIN=USA`. Both are for vehicle type =`TRUCK`. Whenever you replay this, you will see these two analyses. The highlight of this program are the two `LINK TO` statements, ③ and ④. The first statement specifies a link be created in the folder `\linkdir#1`. The link will contain the path of the object `\mylib.quickstart\Univariate#1\ByGroup4#1`. The `FIRST` option specifies that the new link will be placed at the top position in the folder. The `LABEL` option instructs the system to carry over the source object’s label and use it for the link’s label as well.

The second `LINK TO` statement ❸ omits the `LABEL` option, just for the sake of demonstration. The default position, if no positioning argument is given is `LAST`, so the link to the USA Truck output will appear after that for `ASIA`.

The final section shows what the directory looks like. The `DIR ^^` statement ❹ sets the current directory back to the parent of the `LINKDIR#1` folder, namely the root level. The output shows the source and target for each of the links created.

The first link, ❸ has the label of the source object, and the second label is blank. To solidify the concepts of source and target, for this example the path `\linkdir#1\asia_trucks#1` is the *target* and `\mylib.quickstart\Univariate#1\ByGroup4#1` is the *source*. The same is true for the second link, ❹.

Even if you use the `LABEL` option to create your links, you can still change it with the `SETLABEL` statement, as often as you want. Doing so will not affect the link's source data or its label.

Output 10.4: A Document Consisting of Links to Another Document

| A Directory of Links as a Starting Point for a Report on Trucks | | | | |
|--|-------------|--------------------------|-----------------|------------------------|
| Listing of: \Work.Softly\ | | | | |
| Order by: Insertion | | | | |
| Number of levels: 2 | | | | |
| Path | Type | Size in Bytes | Template | Label |
| \linkdir#1 | Dir | | | Truck Comparison |
| \linkdir#1\asia_truck#1 | Link | 32 | | Origin=Asia Type=Truck |
| \linkdir#1\usa_truck#1 | Link | 33 | | |

Program 10.6 shows that the `REPLAY` statement works on links and reproduces the output from the source document.

Program 10.6: Reproducing the Output from the Source Document

```
ods html file="program10_6.html" style=defaultbigger;
proc document name=softly;
replay linkdir; /* or just replay if there is only this one folder */
run;
quit;
```

The output is voluminous, and similar to that in Chapter 1, “Introduction and Quick-Start Guide,” so it is omitted. However, it does illustrate a short, convenient way to replay a specific set of output that interests you. In this case, that output is a comparison of truck data between USA and Asia. Although a similar result could be accomplished with `WHERE=` options, using a link is more compact, so it is easier to specify a group of related output. When it comes time for somebody not familiar with your project to review the output, they can focus quickly on the relevant output because it is all stored together in a conveniently labeled folder. Because they are links, you can create groups of output without having to make a copy of each and every output you want.

Modifying PDF Bookmarks Using Links

Recall from Chapter 6 that some destinations create a bookmark file. In the context of the DOCUMENT, when you replay an object, ODS creates an associated bookmark for that object based on the label path of each output object. The bookmarks are hierarchically organized, so is the bookmark. In order to display bookmarks without the excessive nesting that may make it harder to read, the examples in Chapter 6 showed you that creating a flatter file structure resulted in bookmarks that were easier to read. Those examples used COPY TO statements to accomplish this. Review Output 6.22 to see those results.

Using COPY TO statements raises a question: why create a copy just to print something? In addition, whenever there is more than one of something, you risk versioning problems with your output.

In this section, you will use techniques similar to those in Chapter 6 to create a document of links instead of copies. In particular, you will use graphs from `mylib.firstdoc` to create a PDF of graphs along with bookmarks. This example demonstrates how with links you can see your output replayed in any desired order without modifying the source data, and does so with very little additional storage.

Program 10.7 creates the document of links and prints out the simplified labels to the PDF bookmark file

Program 10.7: Modifying Bookmarks of PDF Files Using Links

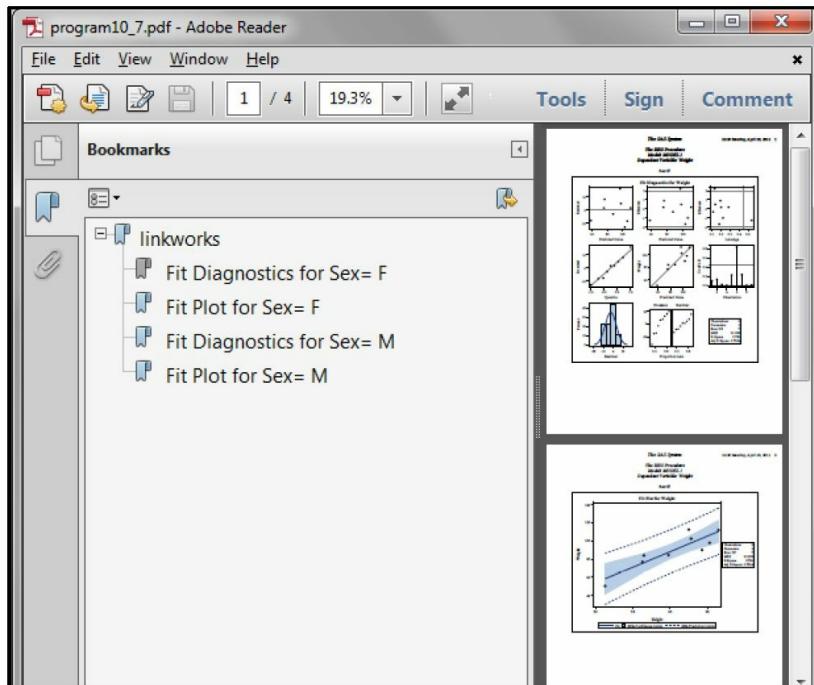
```
proc document name=my.lib.linkonly(write);
  make linkworks;dir linkworks;
*-----;
link \my.lib.firstdoc\Reg#1\By Group 1#1\MODEL1#1\ObswiseStats#1\Weight#1\
DiagnosticPlots#1\DiagnosticsPanel#1 to graph_Fit_F ;
setlabel graph_Fit_F 'Fit Diagnostics for Sex= F';
*-----;
link \my.lib.firstdoc\Reg#1\By Group 1#1\MODEL1#1\ObswiseStats#1\Weight#1\
FitPlot#1 to graph_Fit_F ;
setlabel graph_Fit_F 'Fit Plot for Sex= F';
*-----;
link \my.lib.firstdoc\Reg#1\By Group 2#1\MODEL1#1\ObswiseStats#1\Weight#1\
DiagnosticPlots#1\DiagnosticsPanel#1 to graph_Fit_M ;
setlabel graph_Fit_M 'Fit Diagnostics for Sex= M';
*-----;
link \my.lib.firstdoc\Reg#1\By Group 2#1\MODEL1#1\ObswiseStats#1\Weight#1\
FitPlot#1 to graph_Fit_M ;
setlabel graph_Fit_M 'Fit Plot for Sex= M';
*-----;
replay;run;
quit;
```

Even though the links in this program are long, there are only four of them. If you had more paths to link, you could use CALL EXECUTE techniques, such as those discussed in Chapter 6, “Managing Folders” and Chapter 8, “Exporting to Data Sets,” to create data-driven SAS code from the file of paths. Another approach is to use the LIST statement, and copy and paste the desired paths into your program.

Output 10.5 shows the excerpt from the PDF file. In the excerpt, note the flatter, more readable bookmarks. The bookmarks are readable because the label path has only one level, relative to the current directory.

Thus, links permit you to experiment with different arrangements of your output until you find the one that is most effective, without modifying the source data.

Output 10.5: Shortened Bookmarks Made with Links Instead of Copies



Hard Links

Up until now, you were working with soft, or symbolic links. Recall that a *soft link* is an address of an object. By contrast, a *hard link* is a direct pointer to an output object in the same document. It is indistinguishable from the underlying object. Essentially, a hard link is just an additional name for an object in a file system. (The itemstore that stores ODS objects is a self-contained file system). The critical property of a hard link that makes them useful is the ability to alter output objects by using a more convenient pathname. Why is this useful? Recall that the output objects are usually at the bottom level of a directory tree, and the pathnames of output objects can be quite long. You can use hard links to make it easier to code changes to frequently used output by creating hard links in a more convenient location.

In Program 10.8, demonstrates this. Notice how it becomes easier to manage an object through its shortcut `minirpt#1` at the top level than to navigate through directories.

Program 10.8: Using a Hard Link to Change an Object's Context

```
ods html file='Program10_8.html';
options nobyline;
proc document name=mylib.quickstart;
link \univariate#1\by group 14#1\wheelbase#1\basicmeasures#1
  to minirpt / hard;
setlabel minirpt "usa trucks";
obtitle minirpt#1 "Table 1";
obstitle minirpt#1;
obanote minirpt#1 "This is the Univariate Output for #byval1 #byval2";
replay minirpt#1;
run;
quit;
ods html close;
```

Output 10.6: Results from Program 10.8

Table 1

| Basic Statistical Measures | | |
|----------------------------|-------------|---------------------|
| Location | Variability | |
| Mean | 108.2857 | Std Deviation |
| Median | 109.0000 | Variance |
| Mode | 109.0000 | Range |
| | | Interquartile Range |
| | | 9.00000 |

This is the Univariate Output for USA Wagon

The definition of a hard link stated that a hard link was an additional name for an object. In tangible terms, this means that with a hard link, if you make a change to the target, you change the output object that is the source of the link, and vice versa. Hard links make permanent, internal, changes to the structure of a file system. To keep you from corrupting the item store's file system, the document facility allows only hard links to output objects, as well as document notes created with the NOTE statement, not folders.

Program 10.9 shows another example of how hard links act when they are modified.

```
ods html file='Program10_9.html' style=defaultbigger;
```

```
proc document name=mylib.quickstart;
list; /* default is levels=1 */
run;
quit;
ods html close;
```

The output 10.7 shows that despite the use of a LINK TO statement, the listing shows no objects with type='Link', as it did with the default, soft, links.

Output 10.7: Results from Program 10.9

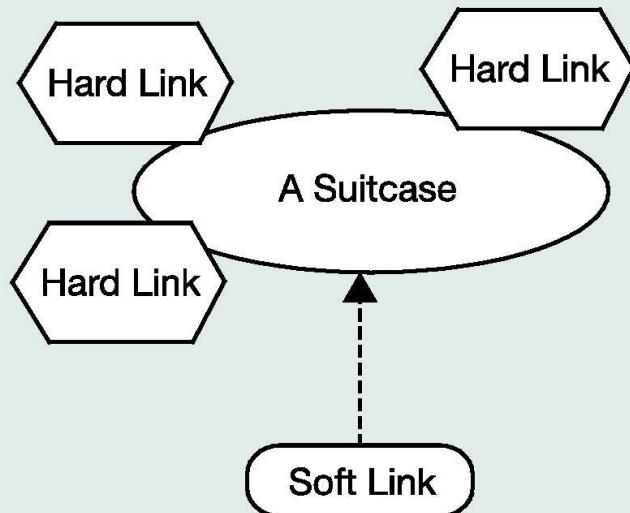
Hard Links are EXACTLY the same as Output Objects

| Listing of: \Mylib.Quickstart\ | |
|--------------------------------|-------|
| Order by: Insertion | |
| Number of levels: 1 | |
| Path | Type |
| Contents#1 | Dir |
| Univariate#1 | Dir |
| Means#1 | Dir |
| Freq#1 | Dir |
| minirpt#1 | Table |

You have seen some concrete examples of both hard and soft links, so it is helpful at this point to discuss in further detail, the difference between hard and soft links. A metaphor that might be helpful for understanding the difference is the situation of keeping your eye on your bag at the airport while traveling with family. See Figure 10.1 for details. As any traveler well knows, many bags look alike, and there is a possibility that somebody may walk off with yours. So, your three kids, serving as the analogy for hard links, agree to keep their hands on your suitcase. So long as at least one hard link exists, one hand is always on the bag and nobody will walk off with it.

Figure 10.1: Hard and Soft Links Explained

Each of the “Hard Links” is like one of your family touching the bag. So long as one hand is touching the bag, it is claimed and nobody will mistakenly take it.



The soft link is a promise only “to keep my eye on the bag.”

Your traveling partner, playing the role of a soft link, agrees to keep his or her eyes on the bag, but only that. The traveling partner might wander farther from the bag (and thereby link to more distant objects or even collections of objects). But there is a price for that freedom—namely a greater risk that (if there are no hard links) that somebody might walk off with the bag even though the partner is carefully watching. In conclusion, you can think of hard links as direct physical connections, and soft links as a promise to retrieve something from elsewhere.

Reasoning through analogy, there are two important restrictions on hard links:

- Hard Links can be made only to output objects, not to folders.
- Hard Links can be made only to objects in the same document.

A summary of the capabilities of hard and soft links is given in the following table:

Table 10.1: Hard Links Compared with Soft Links

| Capability | Hard Link | Soft Link |
|---|-----------|-----------|
| Make changes to titles, subtitles, object notes, and footnotes by using the appropriate statement in the link | YES | NO |
| Link to a path in another document | NO | YES |
| Link to an output object or document note created by the NOTE statement in the same directory | YES | YES |
| Link to a directory | NO | YES |

Accept WHERE clauses to print partial output

Only for variables that are valid for output objects. Not true in general.

NO

Orphaned links possible?

NO (deleting one hard link leaves the others intact)

Link can be orphaned.

The last line deserves some comment. The issue hinges on the word *delete*. To delete an object means precisely to delete *all* hard links to it. Therefore, if there is more than one hard link to an object (and the original object counts as one of the links), the object still exists. This is true even if you delete the first link that you created. All hard links are equal in the mind of the system, so even though the original path is deleted, the object remains. However, it must be addressed by its new path.

To see this for yourself, run Program 10.10. The program creates an object and a hard link to the object. Then, the code deletes the source. However, when replayed, the object is still present, although it can be accessed now only through the path `newer_obj`. Remember, each hard link is a hand on the suitcase. Even though the original hand has moved off, there is still a physical connection to the file, so it can still be managed.

Program 10.10: Deleting the Source Does Not Affect the Target of a Hard Link

```
title 'Deleting the source object might not delete a HARD link';
proc document name=hard_link_test(write);
  copy \mylib.firstdoc\univariate\by group 1\height\moments to orig_object;
  run;

  link orig_object to newer_obj / hard;
  run;

  obtitle newer_obj 'The Table is still present';

  list;
  run;

  delete orig_object;
  run;

  list;
  run;

  replay;
  run;
  quit;
```

Output 10.8 shows that there is still a link to the table, and that the table can still be replayed. In effect, Program 10.10 was really a roundabout way to create a RENAME TO statement, as shown below:

```
RENAME orig_object TO newer_obj
```

This is why we say that a hard link is basically another name for the same object, rather than a shortcut. In fact, most file operations that do not create copies, use hard links (invisibly) to do their work. However, for you, the end user, the important aspect is the ability to submit statements on one object and have your changes appear in another.

Output 10.8: Deleting the Source Does Not Delete the Target for Hard Links

Deleting the source Document does not delete a HARD link

| | |
|-----------------------------------|-------|
| Listing of: \Work.Hard_link_test\ | |
| Order by: Insertion | |
| Number of levels: 1 | |
| Path | Type |
| orig_object#1 | Table |
| newer_link#1 | Table |

| | |
|-----------------------------------|-------|
| Listing of: \Work.Hard_link_test\ | |
| Order by: Insertion | |
| Number of levels: 1 | |
| Path | Type |
| newer_link#1 | Table |

The Table is still present

The UNIVARIATE Procedure
Variable: Height

Sex=F

| Moments | | |
|---------|---|-------------|
| N | 9 | Sum Weights |

Summary

Links permit you to create aliases to objects that you access frequently. This can make it more convenient to manage selections of your favorite outputs without having to search through the directory structure each time you want to modify them. You can also create reports by making folders consisting entirely of links. Such folders are small, but they can communicate their purpose and replay a large amount of related data while requiring very little space to do so.

Soft links are shortcuts to other folders or objects. The source can be an output object, a folder, but not another document. The FOLLOW option in the LIST statement lets you know what will replay when the links are replayed. However, keep in mind that the listed objects are shadows, in the sense that they appear in the listing, but they are not stored in the document that is being listed.

Hard links are not links at all; in listings, they appear as output objects. A hard link can link only from sources that are output objects or document notes, as described in Chapter 7, “Customizing Output,” that reside in the same document. It might help to imagine that if data were stored in a room, a hard link is just a door going directly to that room.

Generally, soft links are preferable and are in line with what most users view as a shortcut. Soft links work across document boundaries. However, the one advantage of hard links is that they let you modify the source. In particular, you can change titles and footnotes, and if the source has a very long path, you can work with it as though it were located at a much shorter path.

Chapter 11: Working with Templates

[Introduction](#)

[What Is a Template?](#)

[Terminology](#)

[Template Store](#)

[Template Search Path](#)

[Template Access Mode](#)

[Updated Document for Examples in This Chapter](#)

[Defining Your Own Template Store](#)

[Template Store Organization](#)

[Exploring the Template System](#)

[Example: Removing Automatic Page Breaks](#)

[Modifying Template Definitions](#)

[Undoing Template Changes](#)

[Templates and the Document](#)

[Identifying Templates in the Document Listing](#)

[The OBTEMPL Statement](#)

[Document Saves a Reference, Not the Actual Template](#)

[Modifying the DOCUMENT Procedure's Templates](#)

[Styles](#)

[Assigning a Custom Template to an Object in a Document](#)

[The Custom Template](#)

[Data Set Used](#)

[Summary](#)

Introduction

This book opened on the premise that if you use the document to store your SAS procedure output, you can replay the output in many different ways without changing your stored output and without rerunning the original analysis. In earlier chapters, this was accomplished by replaying the same output to different destinations. This chapter explains how to change the appearance of your output with ODS table and graph templates (*templates* for short), as well as with ODS style templates (*styles* for short, although they are also templates).

After an introduction to what templates do, this chapter starts with how to set up a personal template store. When the template store is set up, you will explore how to make simple changes to templates that are supplied by SAS. Then, by using the OBTEMPL statement, explore and change templates associated with output from your own documents.

The treatment of the template here focuses on those topics with emphasis on effecting the replay of document output through simple changes to templates. Therefore, some important, but more specialized topics that are related to the TEMPLATE procedure, such as user-designed table templates from scratch, are not covered.

What Is a Template?

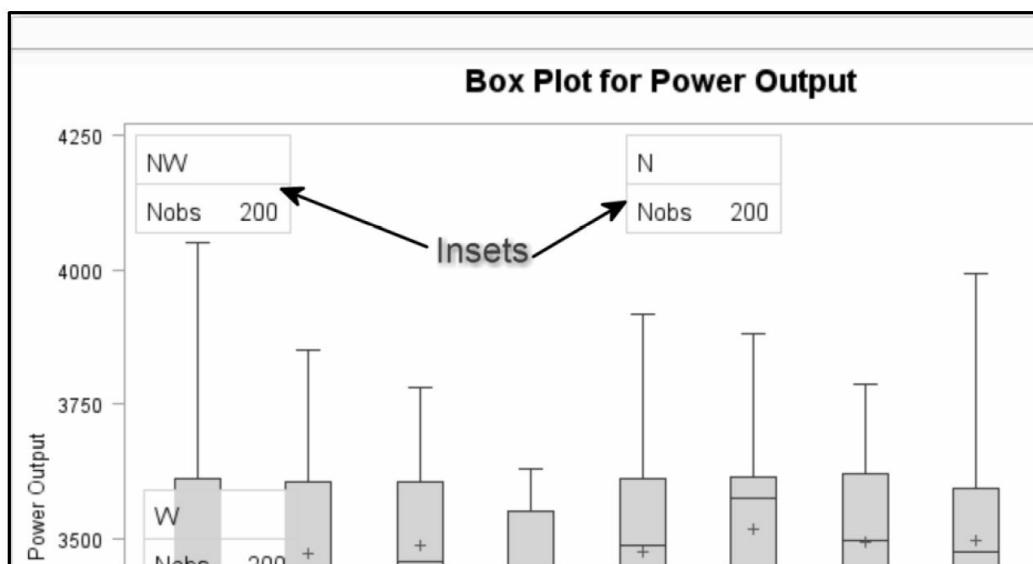
A template is a description of the how an ODS object should be rendered on a page. Templates include columns, headers, styles, and formats of data, as well as overall characteristics such as spacing between cells, table borders, cell borders, and background colors. See the section on column and table attributes in the *SAS 9.3 Output Delivery System User's Guide, Second Edition* for a list of what can be controlled from a template.

This chapter concerns itself with three types of templates. In order of emphasis, they are: table templates, graph templates, and style templates. Table templates, for example, contain a description of the columns and headers. There are also descriptions of attributes such as page breaks and cell spacing. Table templates can contain references to ODS styles. In this chapter, when not otherwise stated, the term *template*, by itself, refers to ODS table templates.

Graph templates contain descriptions of how to build a graph out of graphical elements such as line and bar charts, insets, axes, legends, and other graphical elements. Graph templates are written in the Graph Template Language (GTL), which is a topic too advanced to be discussed here. However, it is important to know that ODS Statistical Graphics are also controlled by templates. Should you learn GTL, you can apply the techniques of this chapter to change the appearance of graphs that are stored in the document by modifying its associated GTL template.

Note: An inset, as mentioned in the previous paragraph, is a text box inside a plot that contains summary information about the data being plotted. Display 11.1 shows an example. To learn more about the elements of Graphics Templates consult Warren Kuhfeld's book, *Statistical Graphics in SAS: An Introduction to the Graph Template Language and the Statistical Graphics Procedures*, as well as the *SAS 9.3 Graph Template Language: User's Guide*.

Display 11.1: Examples of Insets



Terminology

The following terms are necessary for gaining a basic understanding of ODS templates.

Template Store

A template store is a file that contains a collection of template definitions. Like the ODS document, a template store is also an itemstore. Within it, templates are kept in a directory structure.

Template Search Path

The template search path is the list of template stores, in order, in which ODS searches for a specific template name.

Template Access Mode

As with documents, you can access template stores in one of three modes, Read-only, Update, and Write. Their meanings are the same as for document access modes. In order to change a template definition, there must be on the template search path, at least one template store that has update-access mode.

You will see these terms used in examples shortly, after some additional preliminaries are handled, including enhancing our example document with content suited for this chapter.

Updated Document for Examples in This Chapter

Before advancing further into the chapter, Program 11.1 adds some SQL procedure output as well as a SAS data set via the IMPORT TO statement to the document `mylib.firstdoc`.

Program 11.1: ODS Output for SQL Examples

```
ods document name=mylib.firstdoc(update);
proc sql;
  select *
  from sashelp.class
  order by name;
  select *
  from sashelp.cars;
  quit;
ods document close;

/* from Program 9.2 */
data setsused;
label dsname = 'Data Set Name' comment = 'Description';
  length dsname $31 comment $60;
  infile datalines truncover;
  input dsname $$30. comment $$60.;
datalines;
/* &permits spaces in string input */
sashelp.fitness Fitness Data
sashelp.snacks Promotion Effectiveness Data for Snacks
sashelp.classfit Fitness Data for the SASHELP.CLASS folks!
run;

/* add a table */
/* there are TWO spaces between the data elements in the DATALINES stmt. */
proc document name=mylib.firstdoc;
import data=setsused to reftbl / first;
run;
setlabel reftbl 'Example of Imported Table';
quit;
```

Defining Your Own Template Store

The emphasis of this chapter is modifying SAS templates to change the appearance of your output. To do this without disrupting your existing environment, it is necessary to set up a template store to hold your own personal template. Changing a template is a global change, and the changes persist until the template is deleted from the template store, or the template store itself is removed from the path. Template stores are managed by modifying the template search path. The template search path is governed by the ODS PATH statement.

The Syntax of the ODS PATH Statement

```
ODS PATH <(APPEND) | (PREPEND) | (REMOVE)> location(s);  
ODS PATH <(SHOW|VERIFY|RESET)>;
```

To show these statements by example, Program 11.2 creates a template store, `mylib.tempstr`, to hold the results of running the examples in this chapter. Notice that no special statement is needed to create a template. Just put the name in the path, and it will be created if it does not already exist as a template store.

Program 11.2: Defining a New Program Store

```
ods path reset;  
ods path(prepend) mylib.tempstr(update);  
ods path show;
```

This simple program outlines some important concepts surrounding templates. No special statement is needed to create a template store. In this chapter, the examples use Update mode exclusively.

Display 11.2: The Template Search Path Showing New Template Store

```
Current ODS PATH list is:  
  
1. MYLIB.TEMPSTR(UPDATE)  
2. SASUSER.TEMPLAT(UPDATE)  
3. SASHELP.TMPLMST(READ)
```

The template search path is an ordered list. When a program needs a template definition, ODS will search in the order determined by that path until it finds one with the name for which it is searching. The default search path is shown in Display 11.3, and a summary of their meaning and usage is shown in Table 11.1.

To remove your template store (as well as any others), and return to the defaults, use the RESET path argument. This restores the template search path to its default value.

Program 11.3: Restoring Template Arguments

```
Ods path reset;  
Ods path show;
```

Display 11.3: The Default Search Path

Current ODS PATH list is:

1. SASUSER.TEMPLAT(UPDATE)
2. SASHelp.TMPLMST(READ)

The default search path has one updateable template store, `sasuser.templat`, and one Read-only template store, `sashelp.tmplmst`. Until you thoroughly understand the effect that changing your template may have across the system, it is best to keep new template definitions in a template store different from those in Display 11.2. This way, if you make a mistake, you can easily remove the faulty templates from the template search path. Before proceeding with the examples, it is necessary to restore the new template `mylib.tempstr` to the template search path. You can resubmit Program 11.1, or submit the statement in Program 11.4.

Program 11.4: Adding a Template to the Search Path

```
ods path(prepend) mylib.tempstr;  
ods path show;
```

The output is the same as that of Program 11.2.

Now that you know how to create a template-store for your personal templates, and can restore the defaults, you are ready to modify the templates that are used to store SAS procedure output. First, you'll use templates to make a change to the appearance of SQL output.

Template Store Organization

A good practice for template stores is to organize them as shown in Table 11.1:

Table 11.1: Template Store Filenames

| | |
|-----------------------|---|
| SASHELP.TMPLMST | Default SAS templates. This must be in the path, or else ODS cannot produce output. |
| SASUSER.TEMPLAT | Local defaults, maintained by system administrator. Changed primarily at installation to reflect general style preferences for your organization. |
| Other template stores | Application or project-specific templates. |

Exploring the Template System

Much of what is important about changing table or graph templates can be discussed without reference to the Document Facility. The document remains handy, because you can test your template changes on output already stored in a document until you feel comfortable with how the template makes the output appear. In this section, you will learn to make a change to the appearance of a procedure's output by changing a template. The change will be minor, but useful. You will remove the automatic page break between tables for the SQL procedure.

With a few exceptions, each output object that is produced by a SAS procedure is associated with a template, and you can change the appearance of your output (including document replays) by changing its template. This is a four-step process: obtain the name of the desired template, obtain the source code for the template, modify the appropriate columns and attributes, and finally resubmit the code using the TEMPLATE procedure to store the template in an updateable template store.

You have already seen the ODS TRACE statement for determining the path of your output in the document. Recall that it shows the templates as well, as Program 11.5 demonstrates.

Program 11.5: Determining the Name of a Template Using ODS TRACE

```
ods html file="program11_5.html";
ods trace on;
proc document name=mylib.firstdoc;
  replay sql\sql_results;
run;
quit;
ods trace off;
ods html close;
```

The log shown in Display 11.4 shows that the template that is associated with this output is Base.SQL. A template name can be any number of words connected with a period. Expressing the template name hierarchically in this fashion is analogous to the document path, which uses backslashes to separate components. Both determine an item's position in their respective item stores.

Display 11.4: The Template

| Output Added: | |
|---------------|-----------------|
| ----- | |
| Name: | SQL Results |
| Label: | Query Results |
| Template: | Base.SQL |
| Path: | SQL.SQL Results |

To get the template source, run Program 11.6. Then, copy the resulting text to a text editor. The program to extract the source requires that you run the TEMPLATE procedure, which has not been discussed yet. However, the important lines of the examples will be explained. The first statement after the PROC TEMPLATE statement is the SOURCE statement. The SOURCE statement outputs a text file defining the template, shown in the syntax of the TEMPLATE procedure.

Program 11.6: Obtaining Template Source

```
filename example 'sql_template.sas';
proc template;
```

```
source Base.SQL / file=example;
run;
quit;
filename example clear;
```

The TEMPLATE procedure is responsible for managing templates, just as the DOCUMENT procedure is responsible for managing documents. The SOURCE statement sends the source of the template of interest to the file that the file handle `example` points to. The resulting code, which can also be submitted with the TEMPLATE procedure, is shown in Output 11.1. If you are specifically interested in the source of an object stored in a document, you can obtain the template source with the DOCUMENT procedure's OBTEMPL statement. Examples of the OBTEMPL statement will be discussed later in this chapter.

Output 11.1: The Base.SQL Template Source

```
define table Base.SQL ;
notes "PROC SQL Output";
dynamic wmax flowmin flowmax ds;
column obsno character flowcharacter numeric;

define obsno;

define header t;
text _label_;
def_split;
end;
header = t;
space = 2;
format = 6.0;
just = r;
style = RowHeader;
id;
generic;
end;

define character;

define header t;
text _label_;
def_split;
end;
header = t;
width_max = wmax;
space = 2;
just = l;
generic;
end;

define flowcharacter;

define header t;
text _label_;
def_split;
end;
header = t;
width_max = flowmax;
```

```
text _label_;
def_split;
end;
header = t;
width_max = flowmax;
width = flowmin;
space = 2;
just = l;
flow;
generic;
end;

define numeric;

define header t;
text _label_;
def_split;
end;
header = t;
width_max = wmax;
space = 2;
format = best8.3;
just = r;
generic;
end;
double_space = ds;
newpage;
underline;
order_data;
wrap;
wrap_space;
data_format_override;
end;
NOTE: Path 'Base.Sql' is in: SASHELP.TMPLMST.
4 run;
NOTE: PROCEDURE TEMPLATE used (Total process time):
      real time      2.37 seconds
      cpu time       0.71 seconds
```

Discussion of Template Code

A technically detailed and accurate description of the entire template would be rather lengthy. See the references for details about many of these keywords. In particular, you can find more details in Kevin D. Smith's SAS Global Forum paper "PROC TEMPLATE Tables from Scratch," SGF Paper 221-2007. *Output Delivery System: The Basics and Beyond* covers this material in Chapter 10, "Creating Custom Reports from the DATA Step." See the discussion there on "Creating a Custom Table Template with a DATA Step."

The emphasis in this chapter is on reading the templates and making some of the easier cosmetic changes. Don't forget that the Document facility makes learning all phases of ODS easier. You can save your output now and replay with default templates, then replay it again once you've learned more sophisticated techniques.

The most important line in the top section of Output 11.1 is the COLUMN statement. The COLUMN statement lists the variables that the template system expects to see in the table. Further down in the template are the DEFINE COLUMN statements, which contain *attributes* for the column. Attributes are statements that dictate how the column will appear. The first such section defines the attributes for the observation number column. This is followed by the definitions of most columns defined in the COLUMN statement. It is not necessary to have a definition for each column, but most templates do.

Although this brief introduction to templates is not intended as a substitute for a thorough discussion of the topic, the idea of generic columns is important enough to merit a brief mention. Columns in templates can be defined as *generic*. A generic column can store the values of many variables. By contrast, a non-generic column would print only a data table's column if it had the exact same name as the template column. SQL provides two generic columns—one for character variables and one for numeric. The template's internal code figures out which data goes in which generic column.

The template concludes by providing some attributes for the template as a whole. Changing these attributes often provides useful "quick-fix" template changes.

Many of the lines of the table template definition operate by turning on or off some feature of the output, such as:

| | |
|-----------|----------------------|
| Newpage | Wrap_space |
| Underline | Data_format_override |

Other lines are responsible for setting parameters, such as:

| | |
|------------|----------|
| Format=6.0 | Header=T |
|------------|----------|

Now that you have template code in a file, and a basic idea of where to affect changes, the next step is to actually edit the file. To maintain focus on the template changing process, this chapter sticks to fairly innocuous changes. The next change is to remove the automatic page break before each table.

Example: Removing Automatic Page Breaks

The SQL procedure automatically provides page breaks between tables. The automatic page break overrides the page break status implied by the document listing. Program 11.7 shows that there is no page break after the output objects `sql_results#1` or `sql_results#2`.

Program 11.7: SQL Automatically Breaks Pages before Printing Tables

```
ods listing;
ods html file="program11_7.html";
proc document name=mylib.firstdoc;
dir sql;
list / details;
run;
replay sql_results#1(where=(_obs_>15)), /* last few lines */
      sql_results#2(where=(_obs_<5)); /* first few lines */
run;
quit;
ods listing close;
ods html close;
```

Output 11.2 shows the document listing. The page break column states that there is no page break before the object `sql_results #2`, nor a page break after `sql_results #1`. Therefore, based on the document listing, the two `sql_results` objects should print on the same page. Unfortunately, because of the template behavior discussed earlier, they do not.

For the sake of review, the REPLAY statement in Program 11.6 uses the `_OBS_` special variable discussed in Chapter 5, “The REPLAY Statement.” Recall that the `_OBS_` variable is used to restrict which observations of the output object are printed. In this program, it is used to save some space in Output 11.3.

Output 11.2: Listing of SQL#1 Directory Showing Page Breaks Deleted

| Listing of: \Mylib.Firstdoc\SQL#1 | | | | | |
|-----------------------------------|-------|------------------|----------|---------------|------------|
| Order by: Insertion | | | | | |
| Number of levels: 1 | | | | | |
| Path | Type | Size in Bytes | Template | Label | Page Break |
| \SQL#1\SQL_Results#1 | Table | 549 | Base.SQL | Query Results | |
| \SQL#1\SQL_Results#2 | Table | 22231 | Base.SQL | Query Results | |

Output 11.3: The SQL Output Still Has Page Breaks

Page Break Example

| Name | Sex | Age | Height | Weight |
|---------|-----|-----|--------|--------|
| Robert | M | 12 | 64.8 | 128 |
| Ronald | M | 15 | 67 | 133 |
| Thomas | M | 11 | 57.5 | 85 |
| William | M | 15 | 66.5 | 112 |

According to the document's listing, the page break above shouldn't be there

| Make | Model | Type | Origin | DriveTrain | MSRP | Invoice | Engine Size (L) | Cyl |
|-------|----------------|-------|--------|------------|----------|----------|-----------------|-----|
| Acura | MDX | SUV | Asia | All | \$36,945 | \$33,337 | 3.5 | |
| Acura | RSX Type S 2dr | Sedan | Asia | Front | \$23,820 | \$21,761 | 2 | |
| Acura | TSX 4dr | Sedan | Asia | Front | \$26,990 | \$24,647 | 2.4 | |
| Acura | TL 4dr | Sedan | Asia | Front | \$33,195 | \$30,299 | 3.2 | |

Starting with the `Base.SQL` template source from Output 11.1, you can create a new template by editing the line that says `newpage` and changing it to `NEWPAGE=OFF`. Program 11.7 initiates the template procedure to make the change and shows the key features. To save space, Program 11.7 below shows only the relevant lines to change in the template. In addition, you need to make sure the code has a `PROC TEMPLATE` statement at the head of the code, as well as a `RUN` statement at the end.

Program 11.8: Changing an Object's Template

```
proc template;
define table Base.SQL;
  double_space = ds;
<some missing lines>
  underline;
  order_data;
  newpage = OFF; /* <-- this is the changed line */
  wrap_space;
  data_format_override;
end;
run;
```

You can also make changes with the `TEMPLATE` procedure's `EDIT` statement. This can be easier to read if you are making a small number of changes to a template. The change shown in Program 11.8, would be written using the `EDIT` statement, as shown in Program 11.9.

Program 11.9: Changing an Object's Template via the EDIT Statement

```
ods listing;
```

```
proc template;
edit Base.SQL;
  newpage=OFF;
end;

source Base.SQL;
run;
```

However you choose to modify your templates—either directly via the EDIT statement or in two steps by running the SOURCE statement and then editing the resulting file in a text editor—keep in mind where your templates are stored and how changes are managed. This is the subject of the next section.

When you re-run Program 11.6, the page break should be gone. SQL procedure output will no longer automatically generate a new page before each table, provided that the template store MYLIB TEMPSTR is on the template search path.

Modifying Template Definitions

Now that you have seen an example in action, it is time to revisit a conceptual view of the process of locating and applying a template to ODS output. It is important to understand what happens when you change a template's definition. This discussion applies equally to graph templates, table templates, and styles. If your changes don't seem to take effect, or if you want to undo changes, you must know two things before you can act: where these changes reside and how they affect the appearance of ODS output. The ODS PATH statement determines where ODS searches for available templates.

When you edit or change a template provided by SAS, such as `Base.SQL`, you are not changing the default, Read-only copy that is supplied by SAS. Instead, you are *pre-empting other versions further down the search path*. You create a new definition that takes precedence over the original, but the original is still present. Figure 11.1 shows that ODS searches for the definition of `Base.SQL` when the ODS PATH is set to the value shown in Display 11.1.

Figure 11.1: Searching the ODS PATH for a Template

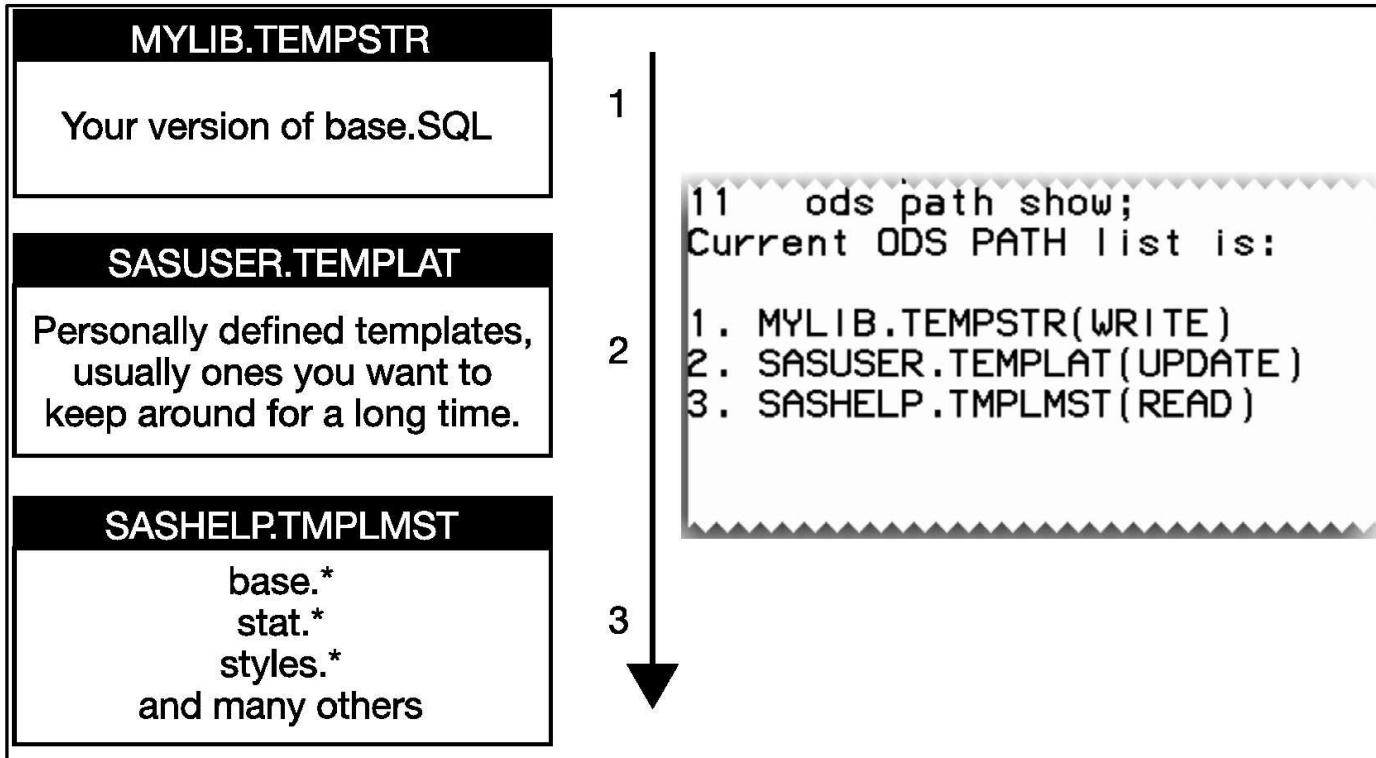


Figure 11.1 demonstrates how ODS uses the template search path. When ODS finds the first copy of the path `Base.SQL` template, it sets that version as the “official” `Base.SQL` table. If no template with the required name, `BASE.SQL`, is found, it moves next to `SASUSER.TEMPLAT`. If no template with the name `BASE.SQL` is found there, it moves to `SASHELP.TMPLMST`.

`SASHELP.TMPLMST` contains the default versions of templates for SAS procedures. It is Read-only, so for all practical purposes, a template is guaranteed to be found for any SAS procedure that uses templates.

The system uses the same search order for modifications as it does for searching. That is, ODS goes to the first updateable (gray) template store on the search path, and writes the new template. An entry to the log tells you where your template was stored, as the log from Program 11.8 states:

```
1077
      end,
NOTE: Overwriting existing template/link: Base.Sql
NOTE: TABLE 'Base.Sql' has been saved to: MYLIB.TEMPSTR
```

Once the template is updated, the SQL procedure will act in accordance with your redefined template. It will not automatically insert new page breaks in between tables for *any* run of the SQL procedure until you either delete the template `BASE.SQL` from the template store `mylib.tempstr` or remove the template store from the template search path.

Undoing Template Changes

You can roll back changes to your templates by either deleting them from a template store, or by removing a template store from the template search path. Program 11.10 shows how to remove the template BASE.SQL from the template store. It is a good practice to delete only individual templates (example: `base.sql`) rather than entire directories (example: `base`)

Program 11.10: Deleting a Template from a Template Store

```
ods path reset; ①  
ods path (prepend) mylib.tempstr(update);  
ods path show;  
proc template; ②  
delete base.sql;  
run;
```

```
2741  delete base.sql;  
NOTE: 'base.sql' has been deleted from: MYLIB.TEMPSTR
```

- ① Although not strictly necessary, the first group of statements re-establishes the template search path before deleting. This provides a level of safety that the right version of the template is indeed being deleted.
- ② The TEMPLATE procedure code deletes the template. Notice that there is no option to specify which template store is affected. Remember, the system searches the template search path in order until it finds an eligible template. The BASE.SQL template is deleted from the template `mylib.tempstr`.

If you want to restore your version of the template, you may re-submit the file that you created in Program 11.6. To remove a template from the template search path, follow the example of Program 11.11.

Program 11.11: Removing a Template from a Template Search Path

```
ods path (remove) mylib.tempstr;  
ods path show;
```

You can make a more comprehensive roll-back of all user-created changes by reverting to the default template search path by submitting the statement:

```
ODS PATH RESET;
```

The statement, `mylib.tempstr`, along with any other user-defined template store, is removed from the path, and any changes will be reverted to the default (assuming you made no changes to `SASUSER.TEMPLAT`).

Templates and the Document

Now that you know the basics of editing templates in general, the remainder of this chapter focuses on templates as they affect the document. For those objects with a template name in the document listing, templates are not stored in the document itself. Only the name of the template is stored there.

You have already seen this phenomenon in action. By changing the BASE.SQL template, you also changed how all SQL procedure output will appear when you replay from your document (as well as all SQL procedure output produced directly by the procedure).

Identifying Templates in the Document Listing

In the DOCUMENT procedure, the LIST statement with the DETAILS option will contain a column showing the name of the template that is associated with each output object that uses templates. Program 11.12 provides such a listing.

Program 11.12: Some Output Objects and their Templates

```
ods html file="program11_12.html";
proc document name=mylib.firstdoc;
list univariate#1\by group#1\heights#1 / details;
run;
quit;
ods html close;
```

Output 11.4 omits columns not related to this discussion, but your output will have additional columns.

Output 11.4: List of Document, Emphasizing Templates

| Listing of: \Mylib.Firstdoc | | | |
|-----------------------------|--|-------|---------------------|
| Order by: Insertion | | | |
| Number of levels: All | | | |
| Obs | Path | Type | Size in Bytes |
| 1 | \Univariate#1 | Dir | |
| 2 | \Univariate#1\ByGroup1#1 | Dir | |
| 3 | \Univariate#1\ByGroup1#1\Height#1 | Dir | |
| 4 | \Univariate#1\ByGroup1#1\Height#1\Moments#1 | Table | 619 |
| 5 | \Univariate#1\ByGroup1#1\Height#1\BasicMeasures#1 | Table | 526 |
| 6 | \Univariate#1\ByGroup1#1\Height#1\TestsForLocation#1 | Table | 578 |
| 7 | \Univariate#1\ByGroup1#1\Height#1\Quantiles#1 | Table | 601 |
| 8 | \Univariate#1\ByGroup1#1\Height#1\ExtremeObs#1 | Table | 412 |

The shading and bolding effects were accomplished using the techniques in the section “Modifying the DOCUMENT Procedure’s templates,” discussed later in this chapter.

The OBTEMPL Statement

So far your template skills consist of creating a new template store and of locating the template that was used to display an output object. Another tool available to you, when working with ODS output stored in the document, is the DOCUMENT procedure's own OBTEMPL statement.

The OBTEMPL statement, new for SAS 9.3, provides another way of printing out a source listing of any object's template, assuming that object has a visible template. The OBTEMPL statement saves you the work of tracking down the template source in the template stores. This is particularly useful when you are running the DOCUMENT procedure interactively. In that case, you can read template sources without having to stop the DOCUMENT procedure.

Syntax of the OBTEMPL Statement

```
OBTEMPL output-object;
```

The template `Base.SQL` that is shown in Program 11.13 displays the familiar SQL template. Notice that the template already reflects that the NEWPAGE=OFF change has been made. Remember, the `Base.SQL` template, as with all others, is not stored in the document itself. The system sees that `Base.SQL` is needed and pulls the first available definition based on the template search path, decompiles the template, and prints its source.

Program 11.13: Extracting the Template for SQL Procedure Output

```
title "The SQL Template";
ods html file='sql_temp1.html';
proc document name=my.lib.firstdoc;
  obtemp1 sql\sql_results;
  run;
quit;
ods html close;
```

The template is now saved in the file that is specified in the ODS HTML statement. Are you wondering how you would know in advance to specify the path `sql\sql_results`? Recall from Chapter 2, “The ODS DOCUMENT Destination,” that the path can be obtained from the ODS TRACE statement when you run the original analysis. Display 11.4 provides such a trace, where the path `SQL\SQL_RESULTS` is mentioned on the fourth line. Another way to know the path of your object is through experience. As you save more output to the document, you'll get to know how your favorite SAS procedures name their output, and consequently in what path those objects will be stored.

One nice feature of the OBTEMPL statement is that it creates ODS output, which can be sent to any destination. Furthermore, the listing includes the PROC TEMPLATE statement, whereas the SOURCE statement from the TEMPLATE procedure creates a listing to the log, and that listing does not have a PROC TEMPLATE statement or a trailing RUN statement.

Document Saves a Reference, Not the Actual Template

The OBTEMPL statement creates a source version of the first version of the template that is named on the template search path. More specifically, it does not extract a template definition from the document itself. You have already learned how to change the BASE.SQL template in order to change the pagination without making any changes to the document itself. This behavior is meant to be a positive feature of ODS. It is worth seeing more examples, so program 11.14 shows how to change output from the UNIVARIATE procedure. The program shows how to display the Quantiles output to display in the SAS format 8.4. (8 characters wide and four decimal places displayed). It also changes the background color for the quantile column.

Program 11.14: Modifying PROC UNIVARIATE Output

```
ods html file=program11_14.html;
proc template;
edit base.univariate.quantiles;
define Estimate;
header = "Estimate";
space = 4;
format=8.4;
style={background=darkgrey foreground=white};
end;
end;
run;
proc document name=mylib.firstdoc(read);
obtemp1 univariate\by group 1\height\quantiles;
replay univariate\by group 1\height\quantiles;
run;
quit;
ods html close;
```

The excerpt from the printout, shown in Output 11.5, still reflects the most recent change to the base.moments.quantiles template. These changes are shown circled below.

Output 11.5: OBTEMPL Always Reflects the Latest Changes Made to Templates

```
define Quantile;
header = "Quantile";
glue = 2;
space = 4;
style = Rowheader;
id;
end;

define Estimate;
header = "Estimate";
format = 8.4;
space = 4;
style = {
color = white
backgroundcolor = darkgrey
};
end;
```

The replay of the quantiles object is shown as Output 11.6.

Output 11.6: Enhanced Quantiles Output

The UNIVARIATE Procedure

Variable: Height

Sex=F

Quantiles (Definition 5)

| Quantile | Estimate |
|------------|----------|
| 100% Max | 66.5000 |
| 99% | 66.5000 |
| 95% | 66.5000 |
| 90% | 66.5000 |
| 75% Q3 | 64.3000 |
| 50% Median | 62.5000 |
| 25% Q1 | 56.5000 |
| 10% | 51.3000 |
| 5% | 51.3000 |
| 1% | 51.3000 |
| 0% Min | 51.3000 |

As one final example of changing a replay through changing a template, Program 11.15 shows the template modification process, this time for output from the MEANS procedure.

Program 11.15: Making and Undoing Template Changes

```
ods html file='program11_15.html';
proc template; ①
  edit base.summary;
  cellstyle mod(_row_,2)=0 as {background=purple foreground=white},
    mod(_row_,2)=1 as data; /* default */
end; ②
run;
* -----
%macro replayit; ③
proc document name=mylib.quickstart;
replay means;
run;
quit;
%mend;
* -----
%replayit
proc template; ④
  delete base.summary;
run;
/* or use :
   ods path(remove) mylib.tempstr; */
%replayit
```

ods html close;

- ① The program begins by redefining the template used by the MEANS procedure, `base.summary`.
- ② You can determine the name of the relevant template by several methods. You can run the ODS TRACE statement (not shown) by referring to the SAS HELP and Documentation: *ODS User's Guide*, or by running an OBTEMPL statement on the object that you want to modify.

```
proc document name=mylib.quickstart;
obtempl means\summary;
run;
quit;
```

The CELLSTYLE AS statement is the standard way to conditionally set the attributes of a template cell. In this template, even numbered rows are now purple. This particular combination is frequently presented in SAS Global Forum papers as how to create rows of alternating color. See, for example, Kevin Smith's paper "The TEMPLATE Procedure Styles: Evolution and Revolution" SUGI-31, Paper 053-31 for more examples of CELLSTYLE AS. See the *Output Delivery System User's Guide* for more examples.

- ③ The document replay is inside a macro because the program needs to execute the code twice: Once after the template change, and again after the template has been deleted.
- ④ If you no longer need to use the modified template, it is good practice to remove it from the path. Therefore, your changes do not hang around after you no longer want them. You can remove its template store from the path with the statement:

```
ods path(remove) mylib.tempstr;
```

After the program deletes the template, notice that the output from the MEANS procedure replay has reverted to the default. Note: If you have redefined the template, MEANS.SUMMARY before running this code, you might have a version of the template in SASUSER.TEMPLAT. In that case, you would see the output revert to that template's output.

Output 11.7 shows the modified replay.

Output 11.7: Changing MEANS Procedure Output

| Origin | Type | N Obs | Variable | Label | N Miss | Mean | Maximum | Minimum |
|--------|--------|-------|-------------|-----------------|--------|-------------|-------------|-------------|
| Asia | SUV | 25 | MSRP | Engine Size (L) | 0 | 29569.00 | 64800.00 | 17163.00 |
| | | | Invoice | | 0 | 26916.48 | 56455.00 | 16949.00 |
| | | | EngineSize | | 0 | 3.4720000 | 5.6000000 | 2.0000000 |
| | | | Cylinders | | 0 | 6.0000000 | 8.0000000 | 4.0000000 |
| | | | Horsepower | | 0 | 214.1600000 | 325.0000000 | 130.0000000 |
| | | | MPG_City | MPG (City) | 0 | 17.3200000 | 22.0000000 | 13.0000000 |
| | | | MPG_Highway | | 0 | 21.6800000 | 27.0000000 | 17.0000000 |
| | | | Weight | Weight (LBS) | 0 | 4108.04 | 5590.00 | 3020.00 |
| | | | Wheelbase | | 0 | 108.0400000 | 129.0000000 | 98.0000000 |
| | | | Length | | 0 | 184.8400000 | 208.0000000 | 163.0000000 |
| | Sedan | 94 | MSRP | Engine Size (L) | 0 | 22763.97 | 55750.00 | 10280.00 |
| | | | Invoice | | 0 | 20788.31 | 48583.00 | 9875.00 |
| | | | EngineSize | | 0 | 2.6478723 | 4.5000000 | 1.5000000 |
| | | | Cylinders | | 0 | 5.0425532 | 8.0000000 | 4.0000000 |
| | | | Horsepower | | 0 | 181.9787234 | 340.0000000 | 103.0000000 |
| | | | MPG_City | MPG (City) | 0 | 22.8404255 | 36.0000000 | 16.0000000 |
| | | | MPG_Highway | | 0 | 29.9680851 | 44.0000000 | 22.0000000 |
| | | | Weight | Weight (LBS) | 0 | 3161.37 | 4802.00 | 2035.00 |
| | | | Wheelbase | | 0 | 105.6489362 | 124.0000000 | 93.0000000 |
| | | | Length | | 0 | 184.0106383 | 204.0000000 | 154.0000000 |
| | Sports | 17 | MSRP | Engine Size (L) | 0 | 32510.65 | 89765.00 | 18739.00 |
| | | | Invoice | | 0 | 29620.94 | 79978.00 | 17101.00 |
| | | | EngineSize | | 0 | 2.4529412 | 4.3000000 | 1.3000000 |

The output for reverting to the default shows that the cell styles that had been altered in the base.summary template have been reverted to the default template.

Output 11.8: Output Restored to Default Appearance

| | | | | | | | | | |
|------|-------|----|-------------|-----------------|----|---|-------------|-------------|-------------|
| Asia | SUV | 25 | MSRP | | 25 | 0 | 29569.00 | 64800.00 | 17163.00 |
| | | | Invoice | | 25 | 0 | 26916.48 | 56455.00 | 16949.00 |
| | | | EngineSize | Engine Size (L) | 25 | 0 | 3.4720000 | 5.6000000 | 2.0000000 |
| | | | Cylinders | | 25 | 0 | 6.0000000 | 8.0000000 | 4.0000000 |
| | | | Horsepower | | 25 | 0 | 214.1600000 | 325.0000000 | 130.0000000 |
| | | | MPG_City | MPG (City) | 25 | 0 | 17.3200000 | 22.0000000 | 13.0000000 |
| | | | MPG_Highway | MPG (Highway) | 25 | 0 | 21.6800000 | 27.0000000 | 17.0000000 |
| | | | Weight | Weight (LBS) | 25 | 0 | 4108.04 | 5590.00 | 3020.00 |
| | | | Wheelbase | Wheelbase (IN) | 25 | 0 | 108.0400000 | 129.0000000 | 98.0000000 |
| | | | Length | Length (IN) | 25 | 0 | 184.8400000 | 208.0000000 | 163.0000000 |
| | Sedan | 94 | MSRP | | 94 | 0 | 22763.97 | 55750.00 | 10280.00 |
| | | | Invoice | | 94 | 0 | 20788.31 | 48583.00 | 9875.00 |
| | | | EngineSize | Engine Size (L) | 94 | 0 | 2.6478723 | 4.5000000 | 1.5000000 |
| | | | Cylinders | | 94 | 0 | 5.0425532 | 8.0000000 | 4.0000000 |
| | | | Horsepower | | 94 | 0 | 181.9787234 | 340.0000000 | 103.0000000 |
| | | | MPG_City | MPG (City) | 94 | 0 | 22.8404255 | 36.0000000 | 16.0000000 |
| | | | MPG_Highway | MPG (Highway) | 94 | 0 | 29.9680851 | 44.0000000 | 22.0000000 |
| | | | Weight | Weight (LBS) | 94 | 0 | 3161.37 | 4802.00 | 2035.00 |
| | | | Wheelbase | Wheelbase (IN) | 94 | 0 | 105.6489362 | 124.0000000 | 93.0000000 |
| | | | Length | Length (IN) | 94 | 0 | 184.0106383 | 204.0000000 | 154.0000000 |

You have now seen two examples of changing the appearance of your replay using the TEMPLATE procedure. You have changed page breaks in the SQL procedure, and changed color on alternating rows. Perhaps these two examples have whetted your appetite for improving your output's appearance by modifying templates.

Modifying the DOCUMENT Procedure's Templates

In addition to changing replayed output, you can also use table templates to modify output from the DOCUMENT procedure's LIST statement. You have already seen several examples of LIST statement output with some columns removed or highlighted, and this section shows how these effects were created.

The relevant template is named `base.document.props`. Program 11.16 shows how to obtain the template source of `base.document.props`. You can copy the contents of the log to a text editor to make the changes, or use an EDIT statement in the TEMPLATE procedure. To keep the programs short, the remainder of examples in this chapter use the EDIT statement for template changes.

Program 11.16: Template Source for DOCUMENT Procedure Listings

```
ods html file='doctempl.html';
proc template;
  source base.document.props;
  run;
ods html close;
```

To suppress printing of a column, *do not delete the COLUMN statement for that column*. This is bad for a number of reasons. The main reason is that it might break ODS code that depends on having a column name present.

Instead, go through each column definition, and change any attributes as needed. To suppress printing of any column in the template, use the PRINT=OFF attribute statement in the column's DEFINE block. The first one is shown for you as Program 11.17.

Program 11.17: Inhibit Display of the OBS Column

```
Proc template;
  edit base.document.props;
    define obs;
      print=off;
    end;
  end;
```

Output 11.9: The Detailed Listing and the New Output Format

Listing of: \Mylib.Firstdoc\

Order by: Insertion

Number of levels: All

| Path | Type | Size in Bytes | Created | |
|---|-------|---------------------|--------------------|------|
| \Univariate#1 | Dir | | 05MAY2012:18:16:24 | 05MA |
| \Univariate#1\ByGroup1#1 | Dir | | 05MAY2012:18:16:24 | 05MA |
| \Univariate#1\ByGroup1#1\Height#1 | Dir | | 05MAY2012:18:16:24 | 05MA |
| \Univariate#1\ByGroup1#1\Height#1\Moments#1 | Table | 619 | 05MAY2012:18:16:24 | 05MA |
| \Univariate#1\ByGroup1#1\Height#1\BasicMeasures#1 | Table | 526 | 05MAY2012:18:16:24 | 05MA |
| \Univariate#1\ByGroup1#1\Height#1\Total#1 | Table | 578 | 05MAY2012:18:16:24 | 05MA |

As a final demonstration of customizing the LIST statement's output, program 11.18 demonstrates how to create document listings similar to those in Output 11.4.

Program 11.18: Highlighting a Column in Document Listing

```

Proc template;
  edit base.document.props;
  define created;
    print=off; /* repeat for all other columns to be turned off */
  end

  define template;
    style = data {
      color = black
      background=lightgrey
      font_weight=bold
    }
  end;
end; /* of edit */
run;
```

Styles

You have already seen how changing the destination, as well as changing the template, can alter the appearance of your document replay. There is one other factor, and you have already seen it in ODS destination statements. The STYLE= option controls fonts, colors, line styles, and other appearance factors that are common to all templates

```
proc template;
  list styles;
run;
```

With the Document facility, you can replay your output in a variety of different styles until you find one just right. Program 11.19 shows the printout of a document object in two different styles, Ocean and Statistical.

Program 11.19: Same Output in Two Different Styles

```
ods html(id=html1) file="program11_19a.html" style=ocean;
ods html(id=html2) file="program11_19b.html" style=statistical;
proc document name=mylib.firstdoc;
  replay univariate/by group 2/height\quantiles
    / dest=(html(id=html1) html(id=html2));
  run;
quit;
ods html(id=html1) close;
ods html(id=html2) close;
```

Program 11.19 demonstrates not only how to print out in different styles, but also how to replay two HTML destinations opened simultaneously. It is not necessary to use the DEST= option, since the REPLAY statement already prints to all open destinations. However, the DEST= option was provided for purposes of review.

The output is omitted, because it is substantially similar to the many quantiles outputs throughout the book. Only the fonts and colors are different.

With styles, as with every other element in ODS, you can always come back and replay your output with more sophistication as you learn more about ODS. The document permits you this flexibility by storing your output in a preformatted state until you are ready to replay it, armed with your newest, greatest, ODS technique

Assigning a Custom Template to an Object in a Document

This section is a little more advanced, because it assumes knowledge of ODS in the DATA step, but this knowledge was not assumed for this book. However, if you know enough to write your own templates, you probably already know how to use ODS in the DATA step. This section shows how to import a table to a document along with a custom template, written from scratch.

The Custom Template

The French-themed template in Program 11.20 is the table template used to illustrate how to import a table with a custom template.

Program 11.20: The Custom Template

```
proc template;
  define table user.tables.tricolor;
    columns blue white red;
    define blue;
      style={background=blue color=white};
      generic;
    end;

    define white;
      style={background=white color=black};
      generic;
    end;

    define red;
      style={background=red color=white};
      generic;
    end;
  end; /* of table definition */
run;
```

Data Set Used

The data set used is YOGURT, a graph of weekly stock prices of a publically traded dairy products company. Each row in the data set is associated with one week of trading. The variables are DT, the start of the trading week, OPEN, the opening price for the week, and CLOSE, the closing price. The actual stock price values are fictitious.

To assign a template to a data set, you must use ODS in the DATA step while the document is open. Take care that the columns used for the PUT _ODS_ statement match the template declared in the FILE PRINT ODS statement, or you might find columns are missing from your output.

Program 11.21: Using ODS in the Data Set with a Custom Template

```
ods document name=import_test(write);
ods html file="ohno.html";
data _null_;
  set yogurt;
  file print ods=(template='user.tables.tricolor'
    columns=(blue=dt (generic=on)
    white=open (generic=on)
    red=close (generic=on)));
put _ods_;
run;
ods document close;

proc document name=import_test;
  dir datastep;
  rename fileprint1#1 ❶ to weekly_prices;
  setlabel weekly_prices 'For first half of 2012 through May 4';
  list / details;
run;
replay;
run;
quit;
ods html close;
```

In addition to importing the data, it is good practice to label your document, and rename the `fileprint` object. The actual pathname for DATATEST procedure output might vary ❶. It will always start with the word ‘`fileprint`’ but the number may vary depending on how many ODS in the DATA step programs you have already run in your session. So, it is important to review your listing. Yours might be named `fileprint<n>#1` if you have previously used ODS in the DATA step prior to running this program. It’s usually wise to quickly rename the `fileprint` object to something that is easier to remember.

The listing is shown as output 11.8. The table has been imported along with the template `user.tables.tricolor`.

Output 11.10: Listing and Replay of a Table with a User-Defined Template

| Listing of: \Work.Import_test\Datastep#1 | | | | | | | |
|--|-------|---------------------|--------------------|--------------------|------------------|----------------------|---|
| Order by: Insertion | | | | | | | |
| Number of levels: 1 | | | | | | | |
| Path | Type | Size in Bytes | Created | Modified | Symbolic Link | Template | Label |
| \Datastep#1 \weekly_prices#1 | Table | 541 | 06MAY2012:21:59:03 | 06MAY2012:21:59:03 | | user.tables.tricolor | For first half of 2012 through May 4 |

| dt | open | close |
|------------|--------|--------|
| 05/04/2012 | 18.447 | 18.07 |
| 04/27/2012 | 18.187 | 18.46 |
| 04/20/2012 | 17.472 | 17.615 |
| 04/13/2012 | 17.329 | 17.199 |
| 04/06/2012 | 18.083 | 17.42 |
| 03/30/2012 | 17.992 | 18.044 |
| 03/23/2012 | 17.888 | 17.654 |
| 03/16/2012 | 18.005 | 18.148 |

If you want to further enhance your template, you can do so, and the replay will automatically reflect your changes, because the template resides in the template store, not in the document itself. When using the template in conjunction with an ODS document, it's not necessary to re-run the ODS in the DATA step. It's simply a matter of changing the template and replaying the output.

If you plan a more elaborate redesign of the table, it may be easier to export the table to a data set using the OUTPUT destination, and then reshape and reformat the table using DATA step code, new templates, new formats, or all of the above.

Program 11.22: Outputs the weekly_prices Object to a Data Set

```
ods output weekly_prices=tmp_prices; ①
proc document name=import_test;
  dir datastep;
  replay / dest=output;
  run;
quit;
ods output close;
```

- ➊ Recall from Chapter 8, “Exporting to Data Sets” that the OUTPUT destination requires an ODS object name and a SAS data set name. Since the object FILEPRINT1#1 was renamed to WEEKLY_PRICES#1, the ODS OUTPUT statement must specify the NAME= option as WEEKLY_PRICES.

For details about other template attributes, see one of the many references to templates in this chapter. The document facility and templates make a powerful pair because the document facility lets you store or defer your style decisions until a later time. Conversely, with the document, it's easy to test changes to a template without having to re-run any of the original analysis that produced the output.

Summary

Although not part of the document facility, table templates give you another tool to modify the appearance of output that is stored in a document. Templates are controlled and created by the TEMPLATE procedure. After defining a temporary *template store* to hold your new templates, you used it to make changes to the output of specific SAS procedures. The DOCUMENT procedure will produce output that reflects these changed templates. Consequently, when you make changes to a template, those changes are reflected in all future replays.

The LIST statement with all of its options can produce very wide output. You can correct this by editing the DOCUMENT procedure's templates for the LIST statement to make the output more concise or to highlight desired columns.

All of these ideas also apply to templates that you develop yourself using ODS in the DATA step.

Appendix: List of Styles Shipped with SAS

[Introduction](#)

[List of Styles Shipped with SAS](#)

[SAS/GRAFH Output and the Document](#)

[Adding Graphics to a Document with SAS/GRAFH Code](#)

[Importing Graphics Directly from Catalogs](#)

[Running the DOCUMENT Procedure in Interactive Mode](#)

[Templates and Styles](#)

[The defaultbigger Style](#)

[The Yogurt Data Set](#)

[SAT Scores Data Set](#)

Introduction

In this appendix programs and tables are labeled based on the referring chapter. This makes it easier for you to return to that chapter in order to reinforce the concepts shown.

If Chapter 1 referred to an item in the appendix, it would be labeled as “1A” in the appendix, the “A” standing for appendix.

List of Styles Shipped with SAS

Output 1A.1 shows the list of styles shipped with SAS. You can use these style names in the STYLE= option for most ODS destinations. You can generate the list of styles using Program 1A.1. If you define your own styles, they will be listed in addition to the SAS styles.

With any style be aware that some output devices may not support some features of that style.

Program 1A.1: Listing All Styles Available

```
ods html file="a1.html" style=big;
title "Styles";
proc template;
  list styles;
run;
ods html close;
```

Output 1A.1: Default Styles Shipped with SAS

| | | |
|---------------|------------------|-------------------|
| Analysis | Astronomy | Banker |
| BarrettsBlue | Curve | Default |
| Dtree | EGDefault | Education |
| Electronics | Festival | FestivalPrinter |
| Gantt | Gears | Harvest |
| HighContrast | Htmlblue | Htmlbluecml |
| Journal | Journal2 | Journal3 |
| Listing | Magnify | Meadow |
| MeadowPrinter | Minimal | Money |
| Monospace | Netdraw | NoFontDefault |
| Normal | NormalPrinter | Ocean |
| Plateau | Printer | Rsvp |
| Rtf | Sasweb | Science |
| Seaside | SeasidePrinter | Sketch |
| Solutions | Statdoc | Statistical |
| Torn | Watercolor | blockPrint |
| fancyPrinter | grayscalePrinter | monochromePrinter |
| sasdocPrinter | | |

SAS/GRAFH Output and the Document

SAS contains several graphics products, each with specific strengths. The examples in the main part of the text focused on two of these: ODS Graphics and ODS Statistical Graphics. These graphics formats, when stored in the document, contain the original analysis data needed to rebuild the graph. This makes it possible to replay the graph, and to create data sets out of the analysis data. One of the original SAS products for graphics, SAS/GRAFH, has not been discussed in conjunction with the document. This situation will now be remedied.

The output from SAS/GRAFH is stored in files called *graphics catalogs*, which consist of a list of graphs called *graphics segments*, or GRSEGs. Each GRSEG represents a single graph. In a typical session this catalog is called WORK.GRSEG, and the novice user need not be aware of its existence until it becomes necessary to save the graphics to a permanent location.

This appendix covers two methods of replaying GRSEG output stored in graphics catalogs. You can use the ODS DOCUMENT statement and submit new SAS/GRAFH code, or you can import selected graphics directly from a previously created graphics catalog with the DOCUMENT procedure's IMPORT TO statement. With both of these methods, it bears repeating that the graphics themselves are not stored in the document. (There are exceptions, which are also mentioned in this section). Exceptions notwithstanding, where SAS/GRAFH output is concerned, only a reference marker is stored in the document itself. Therefore, it is necessary at the time of replay to have access to the underlying graphics catalog.

Adding Graphics to a Document with SAS/GRAFH Code

The CATALOG=graphics-catalog option to the ODS DOCUMENT statement directs the DOCUMENT destination to copy all graphical output produced while it is open to a permanent graphics catalog. This is equivalent to, and more convenient than, using the GOUT=graphics_catalog option on each graphics procedure.

To get started with the examples, you need a permanent graphics catalog. Program 3A.1 creates a simple bar graph and stores it in the graphics catalog sasuser.mygraph.

Program 3A.1: Adding SAS/GRAFH Output to a Document

```
options device=png; ①
ods document name=sasuser.graphdoc(write) cat=sasuser.grstuff; ②
proc gchart data=sashelp.class;
vbar age / discrete;
run;
quit;
ods document close;

proc document name=sasuser.graphdoc;
list / levels=all;
run;
quit;
```

It is not necessary to specify NAME= while the document is open. The document name will default to the name of the open document.

The first line specifies that the graphics will be written to a PNG file ①. The CAT= option ② specifies that all the graphs produced while the document is open will be copied to the graphics catalog sasuser.grstuff.

The listing of the document shows the presence of the graph, as seen in Output 3A.1. If you have been running other SAS/GRAFH examples in the same session, the number following GCHART (but before the '#' sign) may be different. This is a good time to point out that only the numbers following the '#' sign are document sequence numbers. The numbers to the left are part of the name of the graph. SAS/GRAFH names objects as follows: If you run several GCHART procedures in the same session, the first object produced is named GCHART, then GCHART1, GCHART2,...etc. In what follows, the graph we are working with is named GCHART1. Your output will differ only in the number (or lack thereof) after the word 'GCHART'.

Output 3A.1: SAS/GRAFH Output in the Document

| Listing of: \Sasuser.Graphics\ | | |
|--------------------------------|---------------------|-------|
| Order by: Insertion | | |
| Number of levels: 3 | | |
| Obs | Path | Type |
| 1 | \Gchart#1 | Dir |
| 2 | \Gchart#1\GCHART1#1 | Graph |

Although the output type is 'Graph', it is not a self-sufficient graph. That is, you can only replay it

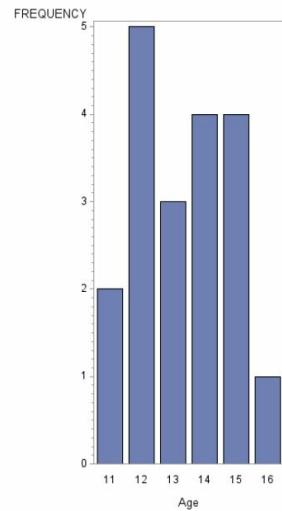
with the help of its graphics catalog. Program A3_2 shows that the document will replay correctly in a new session of SAS so long as the permanent graphics catalog `sasuser.grstuff` is still present.

Program 3A.2: Replying a Graph Stored in a Graphics Catalog

```
proc document name=sasuser.graphdoc;
  replay gchart;
  run;
  ods output close;
  quit;
```

Output 3A.2 shows the replayed graph. You can also replay GRSEGs directly from the catalog with the GREPLAY procedure. However, once rendered as a GRSEG it is not possible to change the appearance of graphical elements such as bar color or spacing unless you re-run the original analysis. Such an exercise is not in the spirit of this book.

Output 3A.2: Replying a Graph Stored in a Catalog



Although it is not possible to change the graph itself, you may add titles, subtitles, and after notes to these graphs, as discussed in Chapter 7, “Customizing Output.” In order to see the titles, you need to use the NOGTITLE option on your ODS *<destination>* statement. You can also use the file management commands from Chapter 6, “Managing Folders.” The final example of this section shows the graph with some annotation. For the value of the macro variable `&objname`, use the name of the current graphics object shown on the bottom line of Output 3A.3.

Program 3A.3: Adding Titles and Notes to a Graph

```
ods html file='replay_cat.html' nogtitle;
%let objname=gchart1;
proc document name=sasuser.graphdoc;
list / levels=all;
run;

dir gchart#1;

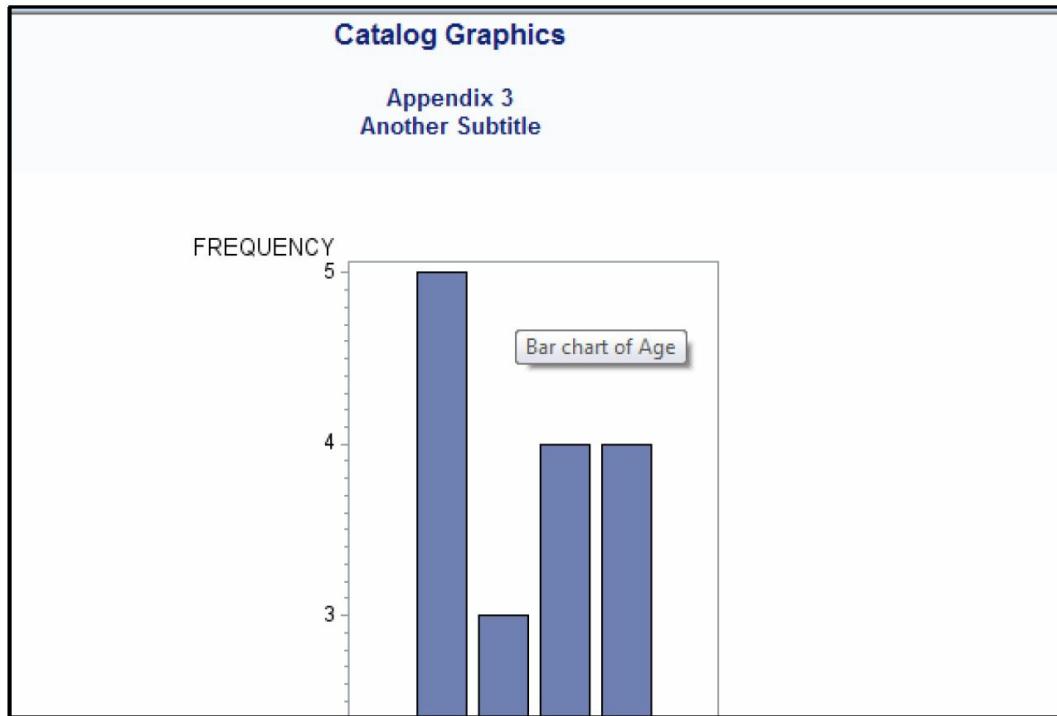
obtitle &objname. "catalog graphics";
obstitle &objname. "appendix 3";
obstitle2 &objname. «another subtitle»;
/* obanote also possible*/
run;

replay &objname.;
run;

quit;
ods html close;
```

The graph, along with its added titles and subtitles is shown in Output 3A.3.

Output 3A.3: A SAS/GRAF Object Replayed with PROC DOCUMENT



The active graphics catalog can be changed, or turned off while the document is still open. The code fragments below show how to do this.

```
ods document catalog=sasuser.different; /* change the active catalog */
ods document catalog=_null_; /* turn off copying */
```

If you do not provide a catalog option, or change the catalog to `_NULL_`, then all graphical output is copied to `WORK.GRSEG`. Although the DOCUMENT destination still saves the output as usual, the REPLAY statement will only work in the current session.

Program 3A.4: Changing the Active Catalog

```
ods html file='null_ctalog.html';
options device=png;
ods document name=sasuser.nullcat(write) cat=sasuser.grstuff;
proc gchart data=sashelp.class;
vbar age / discrete;
run;
ods document cat=_null_;
proc gplot data=sashelp.cars;
plot mpg_highway * mpg_city;
run;
quit;
ods document close;

proc document name=sasuser.nullcat;
list / levels=all;
run;
quit;
ods html close;
```

The important line in this example is the line marked . All graphics catalog entries will no longer be copied to a permanent catalog, so the output from the GPLOT procedure will be saved in `WORK.GRSEG` but nowhere else. When the system is restarted, you will not be able to replay the GPLOT output.

If you want to replay SAS/GRAF output beyond your current session, you must do one of two

things:

- Always use the CATALOG= option to make sure graphs are copied to a permanent catalog.
- Use one of the DEVICE types listed below in the GOPTIONS statement. This will store the graphics in a tabular format within the document, which does not require the presence of a graphics catalog to replay. These are the exceptions referred to in the introduction to this section.

```
goptions device=java|javaimg|activex|actximg;
```

Importing Graphics Directly from Catalogs

If the graphics catalog already exists, you can import its GRSEGs into the document. This gives you some of the advantages of output management that comes with using the DOCUMENT procedure. For example, you can also move the graphs around in the document using the statements in Chapter 6, “Managing Folders”” to create just the right order of graphics, tables, and text to most effectively communicate your ideas.

To import graphs, use the IMPORT TO statement covered in Chapter 9, “Importing Data to the Document.” This section shows the use of the GRSEG option, which was not discussed in that chapter.

Program 3A.5 demonstrates how to import the graph gchart1 from the catalog `sasuser.grstuff` into a document.

Program 3A.5: Importing a Graph from a Graphics Catalog to the Document

```
proc document name=mylib.graphics_example;
  make report_folder; ①
  dir report_folder;

  import grseg=sasuser.grstuff.gchart1 ② to agraphh; ③
  run;

  list \ / levels=all; ④
  run;
  quit;
```

It is not required to save graphs to their own folder; it’s just good practice. When objects are in folders, they are easier to manipulate. The MAKE statement marked ① creates a folder to store the graphics output and the DIR statement makes this directory current. The GRSEG option requires a graphics segment (GRSEG) from a graphics catalog as its argument. This is specified by the three-level catalog entry specification `sasuser.grstuff.gchart1`. ② The first two segments specify the catalog (`sasuser.grstuff`) and the rightmost segment specifies the catalog entry to import (`gchart1`). You can import as many objects as you like, but only one graph is allowed per IMPORT TO statement. The “TO” part of this statement requires a document path. The graph will be kept at the relative path named `agraphh`, a subfolder of `graphics_folder`. ③ The LIST statement produces a listing of the top level (denoted by the single backslash) and uses all levels, denoted by the /LEVELS=ALL option ④. The final document is shown in Output 3A.4. There is no replay shown, since the output would be identical to Output 3A.3.

Output 3A.4: A Document with an Imported Graphics Segment (GRSEG)

The SAS System

Listing of: \Mylib.Graphics_example

Order by: Insertion

Number of levels: All

| Obs | Path | Type |
|-----|-------------------------------|-------|
| 1 | \graphics_folder#1 | Dir |
| 2 | \graphics_folder#1\agegraph#1 | Graph |

In both of the methods described for importing SAS/GRAFH procedure output from a graphics catalog, it bears repeating that the graphics themselves are not stored in the document. You must have the catalog, in this case, `sasuser.grstuff`, defined in your session and loaded with the GRSEGs referred to in the document in order for the document to replay.

Running the DOCUMENT Procedure in Interactive Mode

When running in an interactive environment, you might find it easier to submit the PROC DOCUMENT statement once, and then run the remainder of the examples in interactive mode. This means that you would not need to re-submit the PROC DOCUMENT statement and QUIT statement for each example in this, or the remaining chapters.

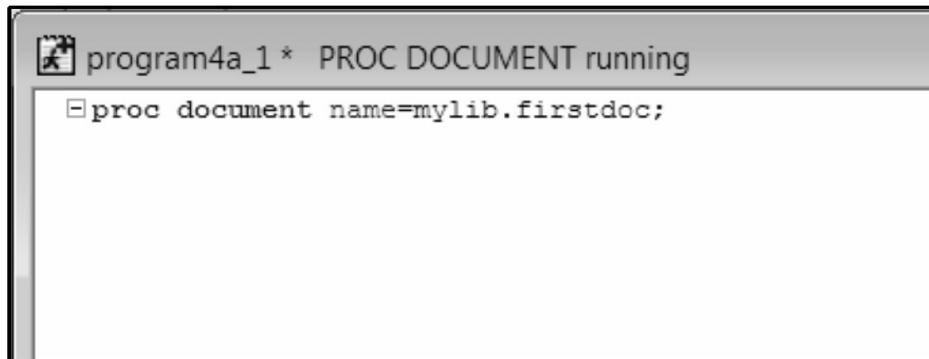
Program 4A.1 shows how to start the DOCUMENT procedure from the windowing environment in SAS. If you are running SAS in a UNIX or Linux environment, you may still have access to a similar windowing environment within that environment's window manager. Consult the SAS companion for your operating system.

Program 4A.1: Starting the DOCUMENT Procedure Interactively

```
proc document name=mylib.quickstart;
```

The result is that the window now informs you that the DOCUMENT procedure is awaiting your command.

Display 4A.1: PROC DOCUMENT Running



A screenshot of a SAS window titled "program4a_1 * PROC DOCUMENT running". The window contains the following text:
proc document name=mylib.firstdoc;

There is no RUN statement. Display 4A.1 shows that the system is now awaiting further commands.

Program 4A.2 shows how submitting a command works when the DOCUMENT procedure is running. The program consists of a single run group. There is no need for the PROC DOCUMENT statement because it is already running.

Program 4A.2: Submit this only if the DOCUMENT Procedure Is already Running

```
ods html style=minimal file='program4A_2_listing.html';
list / details;
run;
ods html close;
```

Program 4A.3 shows how to replay the document. The program also shows that while the DOCUMENT procedure is running, you can open and close additional ODS destinations.

Program 4A.3: Replay A Section of the Document

```
ods html file='program4A_3.html';
replay summaries;
run;
ods html close;
```

While the DOCUMENT procedure is running, you can submit several commands, each sending the output to possibly different ODS destinations.

While it is always important to use RUN statements, when running in an interactive environment,

the procedure will wait in a listening state for the RUN statement.

Finally, Program 4A.4 terminates the procedure.

Program 4A.4: Ending the Document Procedure Interactively

```
quit;
```

When you quit the procedure, you will then see the characteristic message output to the SAS log indicating the procedure has been run. Your ‘Real Time’ will vary of course.

Output 4A.1: Indication that the Procedure is no Longer Running

| |
|--|
| NOTE: PROCEDURE DOCUMENT used (Total process time): |
| real time 16:39.05 |
| cpu time 6.27 seconds |

When you run the DOCUMENT Procedure in interactive mode, you can continue to issue commands to it until you decide to terminate. It is not necessary to constantly repeat the cycle of invoking the DOCUMENT procedure, quitting, and re-invoking it for each set of commands.

Templates and Styles

Chapter 11 showed how changing a template will also affect any document output that refers to this template. The data sets for the demonstration program, as well as for the defaultbigger style that was used in Chapter 11 are included here.

The defaultbigger Style

```
proc template;
  define style defaultbigger;
  parent=styles.htmlblue;
  class fonts
    "Fonts used in the default style" /
    'TitleFont2' = ("<sans-serif>, Helvetica, sans-serif",5,bold italic)
    'TitleFont' = ("<sans-serif>, Helvetica, sans-serif",6,bold italic)
    'StrongFont' = ("<sans-serif>, Helvetica, sans-serif",5,bold)
    'EmphasisFont' = ("<sans-serif>, Helvetica, sans-serif",4,italic)
    'FixedEmphasisFont' = ("<monospace>, Courier, monospace",3,italic)
    'FixedStrongFont' = ("<monospace>, Courier, monospace",3,bold)
    'FixedHeadingFont' = ("<monospace>, Courier, monospace",3)
    'BatchFixedFont' = ("SAS Monospace, <monospace>, Courier, monospace",3)
    'FixedFont' = ("<monospace>, Courier",3)
    'headingEmphasisFont' = ("<sans-serif>, Helvetica,
      sans-serif",5,bold italic)
    'headingFont' = ("<sans-serif>, Helvetica,
      sans-serif",5,bold)
  'docFont' = ("<sans-serif>, Helvetica, sans-serif",5);
  end;
run;
```

The Yogurt Data Set

```
data yogurt;
attrib dt format=mmddyy. informat=mmddyy.;
infile datalines;
input dt open close ;
datalines;
04/29/2012 18.447 18.07
04/22/2012 18.187 18.46
04/15/2012 17.472 17.615
04/08/2012 17.329 17.199
04/01/2012 18.083 17.42
03/25/2012 17.992 18.044
03/18/2012 17.888 17.654
03/11/2012 18.005 18.148
03/04/2012 17.758 17.836
02/26/2012 17.589 17.498
02/19/2012 17.342 17.797
02/12/2012 16.666 17.108
02/05/2012 16.458 16.666
01/29/2012 16.029 16.744
01/22/2012 16.003 16.25
01/15/2012 15.678 16.12
;
run;
```

SAT Scores Data Set

As mentioned in the text, the DATA step code to build work.sat_scores may also be found on the Web by searching the SAS Support Website for ‘Sat Scores Data Set’. When this book went to press, the URL for this code was

<http://support.sas.com/documentation/cdl/en/basess/58133/HTML/default/viewer.htm#a001372196>

Just copy and paste the text to download it. A copy is also available on the author’s Web page at
<http://support.sas.com/publishing/authors/tuchman.html>

```
data sat_scores;
  input Test $ Gender $ Year SATscore @@;
  datalines;
Verbal m 1972 531 Verbal f 1972 529
Verbal m 1973 523 Verbal f 1973 521
Verbal m 1974 524 Verbal f 1974 520
Verbal m 1975 515 Verbal f 1975 509
Verbal m 1976 511 Verbal f 1976 508
Verbal m 1977 509 Verbal f 1977 505
Verbal m 1978 511 Verbal f 1978 503
Verbal m 1979 509 Verbal f 1979 501
Verbal m 1980 506 Verbal f 1980 498
Verbal m 1981 508 Verbal f 1981 496
Verbal m 1982 509 Verbal f 1982 499
Verbal m 1983 508 Verbal f 1983 498
Verbal m 1984 511 Verbal f 1984 498
Verbal m 1985 514 Verbal f 1985 503
Verbal m 1986 515 Verbal f 1986 504
Verbal m 1987 512 Verbal f 1987 502
Verbal m 1988 512 Verbal f 1988 499
Verbal m 1989 510 Verbal f 1989 498
Verbal m 1990 505 Verbal f 1990 496
Verbal m 1991 503 Verbal f 1991 495
Verbal m 1992 504 Verbal f 1992 496
Verbal m 1993 504 Verbal f 1993 497
Verbal m 1994 501 Verbal f 1994 497
Verbal m 1995 505 Verbal f 1995 502
Verbal m 1996 507 Verbal f 1996 503
Verbal m 1997 507 Verbal f 1997 503
Verbal m 1998 509 Verbal f 1998 502
Math m 1972 527 Math f 1972 489
Math m 1973 525 Math f 1973 489
Math m 1974 524 Math f 1974 488
Math m 1975 518 Math f 1975 479
Math m 1976 520 Math f 1976 475
Math m 1977 520 Math f 1977 474
Math m 1978 517 Math f 1978 474
Math m 1979 516 Math f 1979 473
Math m 1980 515 Math f 1980 473
Math m 1981 516 Math f 1981 473
Math m 1982 516 Math f 1982 473
Math m 1983 516 Math f 1983 474
Math m 1984 518 Math f 1984 478
Math m 1985 522 Math f 1985 480
Math m 1986 523 Math f 1986 479
Math m 1987 523 Math f 1987 481
Math m 1988 521 Math f 1988 483
Math m 1989 523 Math f 1989 482
Math m 1990 521 Math f 1990 483
Math m 1991 520 Math f 1991 482
Math m 1992 521 Math f 1992 484
Math m 1993 524 Math f 1993 484
Math m 1994 523 Math f 1994 487
Math m 1995 525 Math f 1995 490
Math m 1996 527 Math f 1996 492
Math m 1997 530 Math f 1997 494
Math m 1998 531 Math f 1998 496
;
```

References

- Haworth, Lauren E., Cynthia L. Zender, and Michele M. Burlew. 2009. *Output Delivery System: The Basics and Beyond*. Cary, NC: SAS Institute Inc.
- Karp, Andrew H. 2007. "A Peek at PROC DOCUMENT." *Proceedings of the SAS Global Forum 2007 Conference*. Cary, NC: SAS Institute Inc. Available at <http://www2.sas.com/proceedings/forum2007/224-2007.pdf>.
- Lawhorn, Bari. 2011. "Let's Give 'Em Something to TOC about: Transforming the Table of Contents of Your PDF File." *Proceedings of the SAS Global Forum 2011 Conference*. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/resources/papers/proceedings11/252-2011.pdf>.
- SAS Institute Inc. 2012. *SAS 9.3 Functions and CALL Routines: Reference*. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. 2012. *SAS 9.3 Language Reference: Concepts*. 2d ed. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/documentation/cdl/en/lrcon/65287/PDF/default/lrcon.pdf>
- SAS Institute Inc., 2012. *SAS 9.3 Output Delivery System User's Guide*. 2d ed. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/documentation/cdl/en/odsug/65308/PDF/default/odsug.pdf>.
- SAS Institute Inc. 2012. Step-by-Step Programming with Base SAS Software. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. 2011. *SAS 9.3 Graph Template Language: User's Guide*. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. 2009. "Creating the Plants Data Set." *SAS 9.2 Output Delivery System: User's Guide*. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/documentation/cdl/en/odsug/61723/HTML/default/viewer.htm#a002915>
- SAS Institute Inc. 2009. SAS Knowledge Base Sample 36505: "Organizing Output Based on BY variable values: PROC DOCUMENT or Macro." Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/kb/36/505.html>.
- Smith, Kevin, D. 2007. "The Output Delivery System (ODS) from Scratch." Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/rnd/base/ods/scratch/index.html>.
- Zender, Cynthia L. 2005. "The Power of TABLE Templates and DATA _NULL." *Proceedings of the Thirtieth Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at <http://www2.sas.com/proceedings/sugi30/088-30.pdf>.

Index

A

absolute pathnames

about 47, 76, 104

DIR statement and 110

access modes

DOCUMENT procedure 51

for templates 237

ODS DOCUMENT statement 15, 40–42

ACTIVEFOOTN option, REPLAY statement (DOCUMENT) 83, 89, 161

ACTIVETITLE option, REPLAY statement (DOCUMENT)

about 83, 155

changing titles with 158

overriding titles with 89

temporary titles and 167

AFTER option, OBPAGE statement (DOCUMENT) 152

AFTER= option (positioning argument) 106

%alldocs macro 141–142

AND logical operator 9

annotations, adding 189

arguments, positioning

about 105–106

importing data sets 213

sequence numbers and 106

attributes (template) 155, 236, 243–244

automatic page breaks 244–246

B

backing up documents 115–116

BASE.SQL template 247–251

BEFORE= option (positioning argument) 106

bookmarks (PDF) 227–228

BY-group variables

building custom subtitles 163–164

BYGROUPS option and 65

listing documents as data sets 193

reordering output 95–96

replaying output using 9

storing in alphabetical order 34
BY line 148, 163
BYGROUPS option, LIST statement (DOCUMENT)
 about 52, 63–64
 BY-group variables and 65
 LEVELS= option 64
 output objects and 64–65
 troubleshooting no output 66–67
BYLINE system option 164
#BYLINE variable 164
#BYVAL variable 163–164
#BYVAR variable 163–164
C
CALL EXECUTE routine 141, 227
caret (^) 110, 225
CATALOG= option, ODS DOCUMENT statement 40
CATALOG procedure 98
 CDATE special document variable 68
 CDATETIME special document variable 68
CELLSTYLE AS statement, TEMPLATE procedure 254
CLOSE action, ODS DOCUMENT statement 4
COLUMN statement, DEFINE TABLE statement (TEMPLATE) 243, 256
concatenating pathnames 105
conditions
 selectively copying based on 119
 subsetting multiple 9
 viewing documents based on 67–68
contains (?) special document variable 9
context
 about 148
 annotating 189
 hard links changing 229
 usage example 148–151
context reference chart 148–150
COPY TO statement, DOCUMENT procedure
 changing ORDER= option and 63
 customizing order of output 109
GUI support 143

LEVEL= option 123–124

managing folders with 115–124

modifying PDF bookmarks using links 227–228

positioning argument 105

WHERE= clause and 67, 106, 117–123

copying

folders and subfolders 115–118

output objects 121

selectively based on conditions 119

CROSSTAB output object

about 12, 62, 64

in ODS documents 147

CSSSTYLE= option, ODS HTML statement 2

CTIME special document variable 68

current directory 47, 130, 225

current document 15, 48, 120

current path 107–108, 120–121

customizing output

context and 148–151

customizing order of 109

document notes 167–174

example document 174–176

output objects in ODS documents 146–147

with OBANOTE statement 165–167

with OBBNOTE statement 165–167

with OBPAGE statement 151–155

with OBSFOOTN statement 161

with OBSTITLE statement 161–165

with OBTITLE statement 156–161

D

data

importing to documents 211–218

preformatted 82

setting up 4

DATA= option, IMPORT statement 212

data sets

combining 178, 186–191

converting ODS documents to 177–178, 192–197

displaying 193
document listing as 193–194
importing 213–214
SASEDOC engine and 197–209
sending output to 179–185

DATASETS procedure 198, 201, 260
debugging WHERE= clauses 71–73

DEFINE COLUMN statement, TEMPLATE procedure 243

DEFINE TABLE statement, TEMPLATE procedure
COLUMN statement 243, 256
NEWPAGE= option 246

DELETE option, OBPAGE statement (DOCUMENT) 152

DELETE statement, DOCUMENT procedure
about 15, 129–132
conditionally viewing documents 67–68
GUI support 143
WHERE= clause and 106

deleting
current directory 130–132
documents 15, 41–42
templates 248–249

DEST= option, REPLAY statement (DOCUMENT)
about 83, 86
directing output with 184–185
replaying in different order 88
replaying to multiple destinations 87

destinations
See specific destinations

DETAILS option, LIST statement (DOCUMENT)
about 11, 52, 59–62
BY-group variables and 65
identifying templates in document listing 249
LEVELS= option and 62
page break information 151–152
SETLABELS statement and 75

DEVICE= system option 200

DIR object 62, 147

DIR= option, ODS DOCUMENT statement 36–40, 109

DIR statement, DOCUMENT procedure

110–111, 225

directories

See folders

DOC statement, DOCUMENT procedure

about 126–127

browsing collections of ODS documents 126–129

LIBRARY= option 127–128

listing all documents in system 140–141

NAME= option 128–129

ODS DOCUMENT statement and 104, 126

DOC_SEQNO data set option 205–208

DOCUMENT destination

about 19

creating folders example 13

determining where output is stored 27

exporting to data sets 191

implementation details for item store 28

naming output 24

ODS TRACE statement 25–27

OUTPUT destination and 183–185, 191

SELECT and EXCLUDE lists 28–30

with ODS 21–24

Document Facility

about 1–3

building graphs from reshaped data 188

changing titles and footnotes 12–13

creating folders 13–15

deleting and overwriting document contents 15

file location 3–4

hard links and 229

managing output 10–12

ODS without 19–21

OUTPUT destination and 178–179

replaying output 5–7

setting up data 4

subsetting output 8–10

document notes

about 167

adding with NOTE statement 170–173

adding with TEXT statement 167–170

annotating 189

inserting 172–173

object notes versus 173–174

viewing with LIST statement 169

DOCUMENT procedure

See also COPY TO statement, DOCUMENT procedure

See also DELETE statement, DOCUMENT procedure

See also LINK TO statement, DOCUMENT procedure

See also LIST statement, DOCUMENT procedure

See also MOVE TO statement, DOCUMENT procedure

See also REPLAY statement, DOCUMENT procedure

about 1

access modes 51

CALL EXECUTE routine and 194–197

checking document assembly 42–43

DIR statement 110–111, 225

DOC statement 104, 126–129, 140–141

HIDE statement 132–136, 143, 171–172

IMPORT TO statement 105

invoking 47–50

LABEL= option 48, 50

MAKE statement 13, 107–109, 143

NAME= option 5–7, 48–49

NOTE statement 143, 170–173

OBANOTE statement 148, 165–167, 173

OBBNOTE statement 148, 165–167

OBFOOTN statement 12, 148, 161

OBPAGE statement 148, 151–155

OBSFOOTN statement 161

OBSTITLE statement 148, 161–165

OBTEMPL statement 242, 250–254

OBTITLE statement 12–13, 148, 156–161

pathnames and 105, 213–214

DOCUMENT procedure (*continued*)

positioning arguments 105–106, 213

QUIT statement 5
reading files 4–5
RENAME TO statement 112–115, 143, 233
role of 23–24
RUN statement and 5, 14
running interactively 50–51
sending output to destinations 5, 178
sequence numbers and 55
SETLABEL statement 50, 75–79, 143, 214
UNHIDE statement 134–135
usage examples 68–74
WHERE= clause and 67–68, 143

documents

See ODS documents

does not contain (!?) special document
variable 8

E

EDIT statement, TEMPLATE procedure 246

engine, SASEDOC

See SASEDOC engine

EQUATION output object

about 12, 62, 64

in ODS documents 147

EXCLUDE list 28–30

EXCLUDED option, ODS TRACE statement 25

exporting to data sets

about 177–178

DOCUMENT destination and 191

OUTPUT destination and 178–197

SASEDOC engine and 197–209

F

FIFO (First-In-First-Out) 194

files

identifying location of 3–4

importing 215–218

reading 4–5

First-In-First-Out (FIFO) 194

FIRST option (positioning argument)

about 106

MAKE statement (DOCUMENT) 107–108

FMTLIB statement, FORMAT procedure 98

folders

accessing output in 198–200

assigning output to specific 36–38

backing up documents to 115–116

copying output objects in 121

copying to subfolders 117–118

creating 13–15

creating copies of 120

creating multiple 38

current directory 47, 130, 225

determining where output is stored 27

hiding 133

inbound style of copying 115–116

label considerations 78–79

listing multiple 59–60

nested 35

outbound style of copying 115–116

renaming 112–114

reordering objects in 124–126

root level directory 47, 118–119

sequence numbers and 53–54

storing output in subfolders 118

WHERE= clause and 106

FOLLOW option, LIST statement (DOCUMENT) 222–223

FOOTNOTE statement 89, 159–160, 165

footnotes

annotating 189

changing 12–13

FORMAT procedure

about 83

FMTLIB statement 98

formats

permanent 98–100

temporary 96–100

formats catalog (needed??) 98

FREQ procedure 5

G

GRAPH output object

 about 12, 62, 64

 in ODS documents 146

 usage example 35

Graph Template Language (GTL) 236

graph templates 236

graphical user interface (GUI) 143

graphics objects 200–202

graphs, re-rendering 203–204

GRSEG= option, IMPORT statement 212

GTL (Graph Template Language) 236

GUI (graphical user interface) 143

H

hard links 228–233

HARD option, LINK TO statement (DOCUMENT) 220, 230

hash sign (#) 27

hidden objects, uncovering 133

HIDE statement, DOCUMENT procedure

 132–136, 143, 171–172

HTML destination

 about 19

 building custom subtitles 163

 managing documents with links 224

 replaying output to 5–6, 87, 89

I

ID= option, ODS HTML statement 89

IMPORT statement

 about 212–213

 DATA= option 212

 GRSEG= option 212

 TEXTFILE= option 212

IMPORT TO statement, DOCUMENT procedure 105

importing data to documents

 about 211–213

 importing data sets 213–214

 importing templates 218

importing text files 215–218

inbound style of copying folders 115–116

item store

about 4, 21–22

implementation details 28

L

LABEL option, **LINK TO** statement (DOCUMENT) 225

LABEL option, **ODS TRACE** statement 25

LABEL= option

DOCUMENT procedure 48, 50

ODS DOCUMENT statement 37, 39–40

LABEL special document variable 68, 74–75

LABELPATH special document variable 8, 68, 74

labels

adding to output 189

building custom 195–196

documents and 76–77

titles versus 77–79

LAST option (positioning argument) 106

LEVELS= option, **COPY TO** statement (DOCUMENT) 123–124

LEVELS= option, **LIST** statement (DOCUMENT)

about 11, 52, 56–60

BYGROUPS option and 64

copying using 123–124

DETAILS option and 62

troubleshooting no output 66–67

uncovering hidden objects 133

WHERE= clause and 71–73

LEVELS= option, **MOVE TO** statement (DOCUMENT) 123–124

LEVELS= option, **REPLAY** statement (DOCUMENT)

about 83, 88

copying using 123–124

usage example 136–140

LIBNAME statement 199

LIBNAME.MEMNAME format 130

LIBRARY= option, **DOC** statement (DOCUMENT) 127–128

LINK object 62, 147

LINK TO statement, DOCUMENT procedure

about 115, 220–221

graphical user interface and 143

HARD option 220, 230

LABEL option 225

managing documents with links 224–226

positioning arguments and 105

links

creating 220–221

document management using 224–228

hard 228–233

modifying PDF bookmarks 227–228

orphaned 223–224

replaying 221–222

soft 220, 222, 228–232

LIST statement, DOCUMENT procedure

See also DETAILS option, LIST statement (DOCUMENT)

See also LEVELS= option, LIST statement (DOCUMENT)

about 51–52

BYGROUPS option 52, 63–67

FOLLOW option 222–223

HIDE statement and 132–133

invoking 52–53

labeling output 189

managing output example 10

modifying PDF bookmarks using links 227

modifying templates 256–258

ORDER= option 52, 62–63

OUTPUT destination converting output from 192–197

sequence numbers 53–56

viewing document contents 34–35

viewing notes with 169

WHERE= clause and 67, 106

listing

all documents in system 140–141

available documents 126–128

multiple folders 59–60

templates 249–250

temporary documents 127

LISTING destination
about 19
checking document assembly 42–43
ODS TRACE statement and 25
LISTING option, ODS TRACE statement 25
logical operators 9
logs
adding documentation to 217–218
capturing 216–217
creating 215–216

M

MAKE statement, DOCUMENT procedure
about 13, 107–109
FIRST option 107–108
GUI support 143
managing folders
about 104
for ODS documents 104–106, 143
usage examples 136–142
WHERE= clause in 106
with COPY TO statement 115–124
with DELETE statement 129–132
with DIR statement 110–111
with DOC statement 126–127
with HIDE statement 132–136
with MAKE statement 107–109
with MOVE TO statement 124–126
with RENAME TO statement 112–115
managing output 10–12
 `_MAX_` special document variable 68, 90
 `_MDATE_` special document variable 68
 `_MDATETIME_` special document variable 68
 `_MIN_` special document variable 68, 90
MOVE TO statement, DOCUMENT procedure
about 5
changing ORDER= option and 63
customizing order of output 109
LEVEL= option 123–124

managing folders with 124–126
positioning argument 105
WHERE= clause and 67, 106
`_MTIME_` special document variable 68
N
`NAME=` option
 DOC statement (DOCUMENT) 128–129
 DOCUMENT procedure 5–7, 48–49
 ODS DOCUMENT statement 4, 32–36
`_NAME_` special document variable 68
naming ODS output 24
nested folders 35
`NEWFILE=` option, ODS HTML statement 84
`NEWPAGE=` option, DEFINE TABLE statement (TEMPLATE) 246
`NOBYLINE` system option 164
`NOOBS` system option 214
`NOT` logical operator 9
`NOTE` object 62, 147
`NOTE` statement, DOCUMENT procedure 143, 170–173
notes
 See document notes
O
`OBANOTE` statement, DOCUMENT procedure
 about 148, 165–167
 document notes compared with 173
 `SHOW` option 165
`OBBNOTE` statement, DOCUMENT procedure
 about 148, 165–167
 `SHOW` option 165
`OBFOOTN` statement, DOCUMENT procedure 12, 148, 161
`OBPAGE` statement, DOCUMENT procedure
 about 151–154
 `AFTER` option 152
 `DELETE` option 152
 document context 148
 ignoring 155
`_OBS_` special document variable 68, 90, 245
observations, excluding 8–9, 245

OBSFOOTN statement, DOCUMENT procedure 161
OBSTITLE statement, DOCUMENT procedure 148, 161–165
OBTEMPL statement, DOCUMENT procedure 242, 250–254
OBTITLE statement, DOCUMENT procedure
 about 12–13, 156–161
 document context 148
 SHOW option 156, 159–160
ODS (Output Delivery System)
 about 1
 with DOCUMENT destination 21–24
 without Document Facility 19–21
ODS _ALL_ CLOSE; statement 85
ODS destination
 about 4, 18–19
 checking document assembly 42–43
 sending output to 20–21, 24
ODS DOCUMENT destination
 See DOCUMENT destination
ODS Document Facility
 See Document Facility
ODS DOCUMENT statement
 about 31–32
 access modes 15, 40–42
 access options in 40–42
 CATALOG= option 40
 CLOSE action 4
 DIR= option 36–40, 109
 DOC statement and 104, 126
 LABEL option 37, 39–40
 NAME= option 4, 32–36
 PATH= option 39
ODS documents
 about 1–2
 adding output to 33–34
 assigning custom templates to objects in 258–261
 backing up 115–116
 browsing collections of 126–129
 checking assembly of 42–43

combining data sets from 186–191
conditionally viewing 67–68
context example 148
converting to data sets 177–178, 192–197
current document 15, 48, 120

ODS documents (*continued*)

deleting 15, 41–42
importing data to 211–218
labeling 76–77
links and 224–228
listing all 140–141
listing available 126–128
listing contents 192–194
managing folders for 104–106, 143
managing via GUI 143
opening 128–129
output objects in 146–147
overwriting 15, 51
preformatted data and 82
reasons to use 24
relabeling 50
renaming 112
reordering BY-grouped output 96
replaying output to all open destinations 84–86
SAS libraries and 211–212
saving output to 4–5
sequence numbers 27, 53–56
special document variables and 67–68, 74
templates and 249–256
updating 40–42
viewing as libraries 201–202
WHERE= clause and 117–123
writing 40–42

ODS EXCLUDE statement 25, 29–30, 73–74

ODS HTML statement

CSSSTYLE= option 2

ID= option 89

NEWFILE= option 84

STYLE= option 84–85
template example 251
ODS name (object) 11, 73
ODS NOPROCTITLE statement 161
ODS output
 See output
ODS OUTPUT statement
 about 179
 CLOSE action 192
 converting documents to data sets 192–197
 SASEDOC engine and 204
 usage example 183–185
ODS PATH statement 238–239, 247
ODS paths
 about 24, 38–40
 absolute pathname 47
 current path 107–108, 120–121
 determining paths to replay 92
 ODS TRACE statement comparison 36
 sequence numbers and 55
ODS PROCTITLE statement 161
ODS SELECT statement 29–30, 73–74
ODS SHOW command 29
ODS styles 25
ODS TEXT statement 167–170
ODS TRACE statement
 about 24–27
 determining names of templates 241, 254
 determining where output is stored 27
 EXCLUDED option 25
 identifying path 251
 LABEL option 25
 LISTING option 25
 ODS path comparison 36
 ON option 36
 ON option, ODS TRACE statement 36
 opening documents 128–129
 operating systems 28

OR logical operator 9

ORDER= option, LIST statement (DOCUMENT) 52, 62–63

orphaned links 223–224

outbound style of copying folders 115–116

output

See also customizing output

See also replaying output

accessing with SASEDOC engine 198–200

adding to documents 33–34

BYGROUPS option and 66–67

determining where stored 27

managing 10–12

reordering BY-grouped 95–96

saving to ODS documents 4–5

sending to DOCUMENT destination 21–24

sending to ODS destination 20–21, 24

sequence numbers in 27

storing at top level 32–35

storing in subfolders 118

subsetting 8–10

two-level hierarchy of 139–140

Output Delivery System

See ODS (Output Delivery System)

Output Delivery System (ODS)

with DOCUMENT destination 21–24

without Document Facility 19–21

OUTPUT destination

about 19, 179

combining data sets from documents 186

converting LIST statement output 192–197

DOCUMENT destination and 183–185, 191

Document Facility and 178–179

listing all documents in system 140–141

replaying to multiple destinations 87

sending output to data sets 179–182

output objects

about 11–12, 24, 62, 64

at top-level 138–139

BYGROUPS option and 64–65

copying 121

hard links to 229

in ODS documents 146–147

LEVELS= option and 147

page breaks and 151–154

renaming 114–115

replaying rows of 91–95

replaying subsets of 90–95

restricting attention to 147

templates and 241

troubleshooting 66–67

WHERE= clause and 106

OVERALL list 29–30

overwrite mode 51

overwriting document contents 15, 51

P

page breaks

output objects and 151–154

removing automatic 244–246

PATH= option, ODS DOCUMENT statement 39

PATH special document variable 68, 74

PDF destination

about 19

building custom subtitles 163

managing documents with links 224

modifying bookmarks using links 227–228

replaying output to 6–7, 87

period (.) 241

permanent formats 98–100

positioning arguments

about 105–106

importing data sets 213

sequence numbers and 106

preformatted data 82

PRINT procedure 199

PRINTTO procedure 215

Q

quick-start guide

- changing titles and footnotes 12–13
- creating folders 13–15
- deleting and overwriting document contents 15
- file location 3–4
- managing output 10–12
- replaying output 5–7
- setting up data 4
- subsetting output 8–10

QUIT statement, DOCUMENT procedure 5

R

re-rendering graphs 203–204

Read access mode

- DOCUMENT procedure 51
 - for templates 237

relabeling documents 50

relative pathnames 76, 105

RENAME TO statement, DOCUMENT procedure 112–115, 143, 233

renaming

- documents 112
- folders 112–114
- output objects 114–115

reordering

- BY-grouped output 95–96
- objects in directories 124–126

REPLAY statement, DOCUMENT procedure

- See also ACTIVETITLE option, REPLAY statement (DOCUMENT)
- about 7, 81–83

ACTIVEFOOTN option 83, 89, 161

building graphs from reshaped data 188

CATALOG= option, ODS DOCUMENT statement and 40

combining data sets 178

DEST= option 83, 86–88, 184–185

extended example folder 150–151

GUI support 143

LEVELS= option 83, 88, 123–124, 136–140

reordering BY-grouped output 95–96

replaying output to all open destinations 84–86

replaying output with temporary formats 96–100
replaying reports 14–15
replaying subsets of output objects 90–95
sending output to OUTPUT destination 183–185, 191
WHERE= clause and 89–90, 106

replaying links 221–222

replaying output

- about 5–7
- changing title and 13
- determining paths to replay 92
- in different order 88
- replaying reports 14–15
- rows of output objects 91–95
- rows of tables 92–95
- to all open destinations 84–86
- to multiple destinations 87
- WHERE= clause and 89–90
- with subsets of output objects 90–95
- with temporary formats 96–100
- without sequence numbers 55–56

REPORT output object

- about 12, 62, 64
- in ODS documents 147

REPORT procedure 14

%restore_list macro 75

root level directory 47, 118–119

RTF destination 19

RUN groups 51, 63, 85

RUN statement

- DOCUMENT procedure and 5, 14

- importance of 192

- REPLAY statement and 85

- RUN group and 50–51, 85

S

SAS item store

- See* item store

SAS library, documents and 211–212

SASEDOC engine

about 197–198
accessing directory of output 198–200
applications of 200–205
sequence numbers and 205–209
`SASHELP.TMPLMST` template 247
`SASUSER.TEMPLAT` template 247, 249, 254
saving output to ODS documents 4–5
`SELECT` list 28–30
`_SEQNO_` special document variable 68
sequence numbers 27, 53–56
positioning arguments and 106
SASEDOC engine and 205–209
`SETLABEL` statement, `DOCUMENT` procedure
about 50, 75–79, 214
graphical user interface and 143
`SGPANEL` procedure 188
`SHOW` option
 `OBANOTE` statement (`DOCUMENT`) 165
 `OBBNOTE` statement (`DOCUMENT`) 165
 `OBTITLE` statement (`DOCUMENT`) 156, 159–160
soft links 220, 222, 228–232
source (links) 220–221, 232–233
`SOURCE` statement, `TEMPLATE` procedure 241–242, 246, 251
special document variables
 about 74
 contains 9
 does not contain 8
 subsetting output objects 90
 underscores and 138
 `WHERE=` clause and 67–68
`SQL` procedure 155
`STYLE=` option, `ODS HTML` statement 84–85
style templates 235–236, 258
subsetting output 8–10
symbolic links 220–221

T
Table of Contents 10–11, 137
TABLE output object

about 11–12, 62, 64

in ODS documents 146

usage example 35

table templates 28, 235–236

tables

page breaks between 244–246

printing charts as 202–203

recalling graphics objects as 201–202

replaying rows of 92–95

saving graphics objects as 201–202

viewing 214

TAGSETS.EXCELXP destination 85

targets (links) 220–221, 232–233

template access mode 237

TEMPLATE procedure

about 168

CELLSTYLE AS statement 254

DEFINE COLUMN statement 243

DEFINE TABLE statement 243, 246, 256

deleting templates 248–249

EDIT statement 246

SOURCE statement 241–242, 246, 251

template search path 237–240, 247–248

template store

about 237

defining 238–240

organization of 240

removing templates from 249

templates

about 236–237, 240–241

access modes supported 237

additional information 243

assigning to objects in documents 258–261

columns in 243–244

deleting 248–249

determining names of 241, 254

graph 236

identifying in document listing 249–250

importing 218
modifying 256–258
modifying definitions 247–248
OBTEMPL statement and 250–254
ODS documents and 249–256
output objects and 241
removing automatic page breaks 244–246
removing form template store 249
restoring arguments 239–240
style 235–236, 258
table 233, 235–236
 undoing changes 248–249, 253–254
temporary documents, listing 127
temporary formats, replaying output with
 96–100
text files, importing 215–218
TEXTFILE= option, IMPORT statement 212
%thin_list macro 75
TITLE statement 89, 159–160, 165
titles
 blank 157–158
 building custom subtitles 163–164
 changing 12–13
 labels versus 77–79
TYPE= data set option 35
TYPE special document variable 68
U
underscore (_) 138
UNHIDE statement, DOCUMENT procedure 134–135
Update access mode
 DOCUMENT procedure 51
 for templates 237
 ODS DOCUMENT statement 15, 40–42
 rerunning code in 114
 updating documents 40–42
W
WHERE= clauses
 common mistake with 117–123

debugging 71–73
GUI support 143
in managing folders 106
ODS names and 73
replaying output and 89–90, 106
special document variables and 90
specifying directories with 47
subsetting output 8–9, 67–68
usage example 70–73
WORK library 127–128
Write access mode
 DOCUMENT procedure 51
 for templates 237
 ODS DOCUMENT statement 15, 40–41
writing documents 40–42
Symbols
 _ (underscore) 138
 . (period) 241
 # (hash sign) 27
 ^ (caret) 110, 225
 !? (does not contain) special document variable 8
 ? (contains) special document variable 9



ACCELERATE
YOUR SAS®
KNOWLEDGE
WITH SAS
BOOKS.

Visit the [SAS Press author pages](#) to learn about our authors and their books, download free chapters, access example code and data, and more.

Browse our [full catalog](#) to find additional books that are just right for you.

Subscribe to our [monthly e-newsletter](#) to get the latest on new books, documentation, and tips—delivered to you.

Browse and search [free SAS documentation](#) sorted by release and by product.

Email us: sasbook@sas.com
Call: 800-727-3228



THE POWER TO KNOW®

© 2012 SAS Institute Inc. All rights reserved. S93144US.0812